Mbug

Interactive Debugger for M.CORE Embedded Processors

User’s Guide

Applies to Mbug Versions 1.6 and later

CHAPTER 1 : INTRODUCTION

Mbug is a flexible software tool enabling evaluation and debugging of both hardware and software developed for M.CORE(tm) 32-bit embedded processors. The comprehensive command set gives the application developer the ability to examine and modify any software accesible hardware resource, assemble and disassemble programs in the target memory, and download programs into the target memory. Additional commands provide the developer the capability to set breakpoints, run programs, trace instructions, and trap exceptions.

This document describes the Mbug features, command set, utilities, platform dependent characteristics, error messages , exception traps, and restrictions. Appendices document platform dependent characteristics as well as providing direction on reconfiguration and extension of Mbug.

1.1 Mbug Features

The Mbug software package provides:

- Modification and display of general purpose, alternate, and control registers
- Assembly and disassembly of M.CORE instructions for modification and display of code
- Single-step trace and continued execution from a specified address
- Modification, display, and movement of system memory
- Setting, displaying, and removing breakpoints
- Extensive on-line help
- Ability to execute user-assembled and/or downloaded software in a controlled environment
- Automatic decompression of compressed s-record files while downloading
- Logging function for generating a transcript of a debugging session

CHAPTER 2 : Mbug OVERVIEW

The following sections describe the minimum required hardware configuration, and the memory model.

2.1 General Hardware Requirements

This Mbug software package can be executed on systems that include a minimum configuration of:

2.1.1 M.CORE CPU

Mbug Version 1.0 supports the M.CORE 1 CPU including implementation specific registers and exceptions.

2.1.2 64k-byte Program Memory

The executable code, read-only data and initialized data sections are stored in the program memory and require approximately 64k bytes of storage.

Mbug program memory may reside in on-chip non-volatile memory, external non-volatile memory, or external RAM. In each case, however, the code expects to gain control of the system immediately out of reset in order to initialize the system to a known state.

2.1.3 12k-byte RAM

System RAM is used for initialized data, uninitialized data, and serial I/O buffers. Initialized data is copied from ROM to RAM during initialization.

In the standard configuration, RAM requirements total approximately 12k bytes. Of this total, slightly over 8k bytes are allocated to the serial I/O buffers. Less RAM could be allocated, as long as hardware handshaking is enabled on the serial port connected to the host computer for downloads and transparent mode operation. Changing the size of the serial I/O buffers requires a re-compilation of the monitor program.

2.1.4 Serial I/O Ports

Mbug requires a minimum of one serial port for proper operation, although two are required for some applications. Version 1.0 of the software has been configured to use the two serial ports on the MMC2001 chip. With different I/O drivers, any asynchronous serial peripheral could be used. Changing the serial port I/O drivers requires a re-compilation of the monitor program.

2.2 Mbug Memory Map

The Mbug memory map is hardware (silicon and board) dependent. A detailed diagram can be found in the appendices.

CHAPTER 3 : Mbug COMMANDS

This chapter describes the Mbug command line syntax and command functionality.

3.1 Command Line Syntax

Commands are entered with the following syntax. Multiple commands can be entered on the same line if separated by a semicolon, “;”.

```
Mbug >>[<delimiter>]<command><delimiter[<parameters>]<command_terminator>

<delimiter> := {\ | ,| \t}

<command> := valid command name

<parameters> := <parameter> [<delimiter> <parameters>]

<parameter> := valid parameter for <command>

<command_terminator> := {\n | \r | ;}
```

3.2 Command Parameter Number Syntax

All input numbers to the command line are hexadecimal. a leading ”0x” is not required, nor is it allowed. Leading zeros are optional. The hexadecimal input restriction does not apply to the single line assembler. See CHAPTER 4 : SINGLE LINE ASSEMBLER/DISASSEMBLER for a description of the single line assembler syntax.

Some commands support address ranges. Ranges are specified on the command line as two addresses separated by a dash, “-”. Open-ended address ranges are also supported, and are specified as a single address followed by a plus, “+”.

3.3 Command Parameter Register Syntax

The general purpose registers are specified as r*x*, where *x* is an integer from 0 to 15. Alternate registers are specified by a*x*, where *x* is an integer from 0 to 15. Control registers are specified as cr*x*, where *x* is an integer from 0 to 31. Since not all control registers are implemented on all M.CORE implementations, some numbers in the range may not be valid. For instance, M.CORE 1 only implements the first 13 registers so only 0-12 are valid. Control registers may also be referenced by their standard mnemonic.

The single letter “r” can be used as a shorthand notation to reference the entire general purpose register set. Likewise, the letter “a” can be used to reference the entire alternate register set. A shorthand specifier for the control register family is not provided since they are not considered a family in the same sense as the general and alternate registers.

3.4 Terminating Interactive Commands

The single letter “x” will terminate a command that is operating in interactive mode.

3.5 Command Detailed Descriptions

Detailed descriptions are supplied for each of the Mbug commands. These descriptions include the command name, command line parameter options, a detailed description of command operation, and an example(s) of the command usage. User-entered commands appear in boldface throughout this chapter.

. (period) .

repeat last command

.

Typing a period will repeat the last command entered.

Example:

```
Mbug >> trace 2100
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002104  st.w r13,(r01,0)
Mbug >> trace +
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002106 addu r13,r01
Mbug >> .
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002108 mtcr r13,ss0
Mbug >>
```

about

Mbug Version Information

about

The version information for the current implementation of the Mbug monitor will be displayed on the terminal.

Example:

```
Mbug >> about
M     M
MM   MM
M M M M
M  M  M
M     M bug

Version: 1.6

Copyright Motorola Inc., 1993, 1994, 1995, 1997, 1998

Mbug >>
```

as

Single-Line Assembler

as address

as start+

as start - end

The single-line assembler for the Mbug system will display the contents of memory at the given location and enter interactive mode. The user will be queried for a valid mnemonic and operands which will be assembled into a valid opcode and stored at that memory location. A location can be left unmodified by typing <return> to pass over it.

The “plus” form of the command will allow the user to start assembling code at a given start location and continue through increasing addresses until terminated by the user. The “range” version will start at the first address location and automatically terminate at the given end address.

At any point “x” can be entered as a mnemonic and **as** will terminate and return the user to the Mbug prompt.

Details of the Assembler syntax are documented in CHAPTER 4 : SINGLE LINE ASSEMBLER/DISASSEMBLER.

Examples:

```
Mbug>>as 0+
0x00000000 0x0000 bkpt          >> movi R2,31
0x00000002 0x0000 bkpt          >> addi r1,r2
0x00000004 0x0000 bkpt          >> ld r2,(r1,0)
0x00000006 0x0000 bkpt          >> bkpt
0x00000008 0x0000 bkpt          >> x
Mbug >>
```

bd

Breakpoint Delete

brdel address

brdel will delete a breakpoint that was previously set at a specified address in memory. This command will remove one breakpoint at a time. If several breakpoints must be deleted, this command must be executed once for each address.

This command has been made obselete by the "nobr" command introduced in Version 1.5 and may be removed in later releases.

Examples:

```
Mbug >>brset 3400
breakpoint set at 0x00003400
Mbug >>brset 3420
breakpoint set at 0x00003420
Mbug >>brlist
Current Breakpoint List:
        0x00003400
        0x00003420
Mbug >>brdel 3420
breakpoint deleted from 0x00003420
Mbug >>brlist
Current Breakpoint List:
        0x00003400
Mbug >>
```

bl

List Breakpoints

bl

The breakpoints that are currently set will be displayed on the terminal.

This command has been made obselete by the “br” command introduced in Version 1.5 and may be removed in later releases.

Examples:

```
Mbug >>brlist
Current Breakpoint List:
Mbug >>brset 3400
breakpoint set at 0x00003400
Mbug >>brlist
Current Breakpoint List:
        0x00003400
Mbug >>brset 3420
breakpoint set at 0x00003420
Mbug >>brlist
Current Breakpoint List:
        0x00003400
        0x00003420
Mbug >>
```

br

Set a Breakpoint

br **address**

br

br will set a breakpoint at a specified address in memory. Breakpoint set will not remove a breakpoint from an address if a breakpoint already exists there.

This command will set one breakpoint at a time. If several breakpoints must be set, this command must be executed once for each address. A maximum of 20 breakpoints can be set in the system.

br will list all currently set breakpoints when invoked without the address argument.

Examples:

```
Mbug >>br
Current Breakpoint List:
Mbug >>br 3400
breakpoint set at 0x00003400
Mbug >>br
Current Breakpoint List:
        0x00003400
Mbug >>br 3420
breakpoint set at 0x00003420
Mbug >>br
Current Breakpoint List:
        0x00003400
        0x00003420
Mbug >>
```

bs bs

Set a Breakpoint

brset address

brset will set a breakpoint at a specified address in memory. Breakpoint set will not remove a breakpoint from an address if a breakpoint already exists there.

This command will set one breakpoint at a time. If several breakpoints must be set, this command must be executed once for each address. A maximum of 20 breakpoints can be set in the system.

This command has been made obselete by the “br” command introduced in Version 1.5 and may be removed in later releases.

Examples:

```
Mbug >>brlist
Current Breakpoint List:
Mbug >>brset 3400
breakpoint set at 0x00003400
Mbug >>brlist
Current Breakpoint List:
        0x00003400
Mbug >>brset 3420
breakpoint set at 0x00003420
Mbug >>brlist
Current Breakpoint List:
        0x00003400
        0x00003420
Mbug >>
```

da

Define Alias

da

This command will allow the user to define an alias to a list of commands (separated by a semicolon). Once the alias has been defined, Run Alias (**ra)** can be used instead of retyping the list of commands. Only one alias may be set at a time, and using Define Alias a second time will overwrite the previously aliased command list.

Below is an example of using an alias to define a combination of registers and memory to be displayed each time the alias is run.

Example:

```
Mbug >>da
Current alias definition:

New alias : rd r a epsr epc; md 1000
Alias defined as : rd r a epsr epc; md 1000

Mbug >>ra
r00:  0x00000000        r01:  0x0000ff00
r02:  0x00000000        r03:  0x00000000
r04:  0x00000000        r05:  0x00000000
r06:  0x00000000        r07:  0x00000000
r08:  0x00000000        r09:  0x00000000
r10:  0x00000000        r11:  0x00000000
r12:  0x00000000        r13:  0x00000000
r14:  0x00000000        r15:  0x00000000

a00:  0x00000000        a01:  0x0000ff00
a02:  0x00000000        a03:  0x00000000
a04:  0x00000000        a05:  0x00000000
a06:  0x00000000        a07:  0x00000000
a08:  0x00000000        a09:  0x00000000
a10:  0x00000000        a11:  0x00000000
a12:  0x00000000        a13:  0x00000000
a14:  0x00000000        a15:  0x00000000

epsr : 0x00000000
epc : 0x00000000
0x00001000      00000000 00000000 00000000 00000000     ................
Mbug >>
```

dl

Download Data From the Host

dl

This instruction provides the ability to receive data from the host serial port. The data can be received in two formats: S-Records or compressed S-Records. The firmware will automatically detect the format and decompress the data if necessary. The data which is downloaded will be placed in memory locations specified by the input file. See Chapter ?? for more information on compression and decompression.

The S-record checksums are ignored during downloads.

Example :

```
Mbug >> tm
```

(On the host machine (Unix):)

```
Unix $ ls
hello.c
hello.o
hello.srec
a.out
Unix $ cat hello.srec (<ctrl>-a)
```

(On the target board running Mbug:)

```
Mbug >> dl
Download Complete.
Mbug >>
```

Example :

```
Mbug >> dl
```

(On the host machine (PC):)

Use the terminal emulator's pull-down menu's to initiate transfer of the S-record file.Transfer mode should be text or ASCII for S-Records.

(On the target board running Mbug:)

```
Download Complete.
Mbug >>
```

ds

Disassembler

ds **address**

ds **start +**

ds **start - end**

The disassembler for the Mbug system displays the contents of memory at the given address. The contents are shown in hexadecimal opcode format as well as in M.CORE assembly instruction format.

If the “plus” form is used, the command goes into interactive mode and will continue reading and disassembling until terminated by the user. If the “range” form is used, the command will continue reading and disassembling for each inclusive address in the range specified.

The different forms can be combined in a single command by separating the forms with a comma or whitespace. This will display multiple disassembled portions of the memory space with one command.

Examples:

```
Mbug >>ds 0,4
0x00000000 0x61f2 movi    r2,0x1f
0x00000004 0x8201 ld      r1,(r2,0x0)
Mbug >>ds 0
0x00000000 0x61f2 movi    r2,0x1f
Mbug >>ds 0+
0x00000000 0x61f2 movi    r2,0x1f
0x00000002 0x2011 addi    r1,0x2
0x00000004 0x8201 ld      r1,(r2,0x0)
0x00000006 0x0000 bkpt
0x00000008 0x0000 bkpt
0x0000000a 0x0000 bkpt
0x0000000c 0x0000 bkpt
0x0000000e 0x0000 bkpt
0x00000010 0x0000 bkpt
0x00000012 0x0000 bkpt
0x00000014 0x0000 bkpt
0x00000016 0x0000 bkpt
0x00000018 0x0000 bkpt
0x0000001a 0x0000 bkpt
0x0000001c 0x0000 bkpt
0x0000001e 0x0000 bkpt
x to quit, anything else to continue >> x
Mbug >>ds 2-4
0x00000002 0x2011 addi    r1,0x2
0x00000004 0x8201 ld      r1,(r2,0x0)
Mbug >>
```

go

Execute User Code

go **address**

go +

This command allows the user to initiate execution of code starting at a given address. An rte instruction is used to transfer control from the monitor to user code. The **go** command copies the start address into the EPC (Exception Program Counter) register (if supplied). The rte instruction then loads the PC from the EPC and the PSR from the EPSR (Exception Processor Status Register). The user must have previously written EPSR to the desired value.

The “plus” form will continue execution at the address already contained in the EPC register. This is useful for continuing where a breakpoint had previously stopped execution.

Control returns to the monitor whenever the user program encounters an exception, provided the user program has not rewritten the vector table or Vector Base Register (VBR). The address of the instruction currently executing will be written to the screen as well as the disassembled instruction at that address.

Examples:

```
Mbug >> ds 4de-4ec
0x000004de 0x6004 movi    r4,0x0
0x000004e0 0x6325 movi    r5,0x32
0x000004e2 0x8604 ld      r4,(r6,0x0)
0x000004e4 0x1c56 addu    r6,r5
0x000004e6 0x9604 st      r4,(r6,0x0)
0x000004e8 0x2003 addi    r3,0x1
0x000004ea 0x0f23 cmpne   r3,r2
0x000004ec 0xeff8 bf      0x000004e2
Mbug >> br 4e6
breakpoint set at 0x000004e6
Mbug >> go 4de
A Program exception has occurred.
Breakpoint Encountered:
Current Instruction Pointer: 0x000004e6 st      r4,(r6,0x0)
Mbug >> go +
A Run Mode or Trace exception has occurred.
A Program exception has occurred.
Breakpoint Encountered:
Current Instruction Pointer: 0x000004e6 st      r4,(r6,0x0)
Mbug >>
```

help he

Help on Mbug Commands

help **command**

he **command**

This provides information on the commands implemented by Mbug. For a full list of commands see **menu**.

Examples:

```
Mbug >>help go
GO:
===
Syntax: go <address>
        go +

Description:

This command allows the user to execute user code
starting at <address>. If the "+" argument is used, then
the execution starting point is defined by the contents of
bits 31-1 of the user's EPC.

Note that if a breakpoint is encountered, then entering
"go +" will allow execution to continue where the
breakpoint stopped it.
Mbug >>help md
MEMORY DISPLAY:
==============
Syntax: md <address>[{-<address>,+}]
        mdh <address>[{-<address>,+}]
        mdb <address>[{-<address>,+}]

Description:

Displays data stored in the given memory location(s).
The display will always start aligned to the 16-byte boundary
containing the supplied starting address. Addresses must be aligned
to the size of the displayed data. The width of the memory accesses
corresponds to the width of the displayed data. 'md' displays four
32-bit words per line formatted as 8 hex digits each; 'mdh' displays 8
16-bit halfwords per line formatted as 4 hex digits each; and, 'mdb'
displays 16 8-bit bytes per lineformatted as 2 hex digits each.
The ASCII representation of the data is also printed in each case.

If the specified address range exceeds 64 bytes, the command will enter
an interactive mode, pausing after each four lines of displayed data. At
each pause the user can select whether to continue or exit the command early.
A <cr> signals the next four lines to be displayed. This sequence repeats
until the entire memory range has been displayed, or the user enters an 'x'
at the prompt
Mbug >>
```

log

Toggles Transcript Logging

log

This command provides the capability to log a debug session. The command toggles the logging function. When logging is enabled, all characters sent to the terminal will be echoed to the host port.

The log function is automatically disabled if Set Input (**si**) has been used to indicate that all I/O (terminal interaction as well as downloads) will go through the keyboard serial port.

Example:

```
Mbug >> log
You are enabling logging! After this message all input and output to your
terminal will be mirrored out to the host port. Now would be a time to open an
editor on the host and get into insert mode
Mbug >> log
Logging disabled!
Mbug >>
```

md

Display Memory Word

md address

md start +

md start - end

This command will display memory as 4 words per line of output. If the “range” form is used and it exceeds 16 words then memory display will enter interactive mode. In interactive mode, the user needs to type <return> to continue or “x” to exit. The start address is normalized to the previous quad-word boundary. Likewise, the ending address is normalized to the next quad-word boundary. For example, if the start address was 0x00000104 then the first memory address to be displayed would be 0x00000100. If the end address was 0x00000104 then the last location to be displayed would be 0x0000010C. All address arguments must be word aligned.

The “plus” form will cause the command to enter interactive mode and display memory in 16-word blocks from a given start address and continue until the user types “x”.

Note that the above parameter forms can be combined by separating the forms with a comma or whitespace. This will display multiple portions of the memory space with one command.

Examples:

```
Mbug >> md 50100,50200
0x00050100  00000041 00000042 00000043 00000044         ...A...B...C...D
0x00050200  00000000 00000000 00000000 00000000         ................
Mbug >> md 50100-50130
0x00050100  00000041 00000042 00000043 00000044         ...A...B...C...D
0x00050110  00000045 00000046 00000047 00000048         ...E...F...G...H
0x00050120  00000000 00000000 00000000 00000000         ................
0x00050130  00000000 00000000 00000000 00000000         ................
Mbug >> md 50250+
0x00050250  40000040 00000000 00000000 00000000         @..@............
0x00050260  00000000 00000000 00000000 00000000         ................
0x00050270  00000000 00000000 00000000 00000000         ................
0x00050280  00000000 00000000 00000000 24002400         ............$.$.
md >>
0x00050290  00000000 00000000 00000000 00000000         ................
0x000502a0  2a2a2a2a 00000000 00000000 00000000         ****............
0x000502b0  00000000 00000000 00000000 00000000         ................
0x000502c0  00000000 00000000 00000000 00000000         ................
md >>x
Mbug >>
```

mdh

mdh

Display Memory Halfword

mdh address

mdh start +

mdh start - end

This command will display memory as 8 halfwords per line of output. If the “range” form is used and it exceeds 32 halfwords then memory display will enter interactive mode. In interactive mode, the user needs to type <return> to continue or “x” to exit. The start address is normalized to the previous quad-word boundary. Likewise, the ending address is normalized to the next quad-word boundary. For example, if the start address was 0x00000104 then the first memory address to be displayed would be 0x00000100. If the end address was 0x00000104 then the last location to be displayed would be 0x0000010C. All address arguments must be halfword aligned.

The “plus” form will cause the command to enter interactive mode and display memory in 16-word blocks from a given start address and continue until the user types “x”.

Note that the above parameter forms can be combined by separating the forms with a comma or whitespace. This will display multiple portions of the memory space with one command.

Examples:

```
Mbug >>mdh 100,200
0x00000100   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00000200   0000 0000 0000 0000 0000 0000 0000 0000    ................
Mbug >>mdh 1000-1030
0x00001000   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001010   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001020   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001030   0000 0000 0000 0000 0000 0000 0000 0000    ................
Mbug >>mdh 1000-1080
0x00001000   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001010   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001020   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001030   0000 0000 0000 0000 0000 0000 0000 0000    ................
mdh >>
0x00001040   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001050   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001060   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001070   0000 0000 0000 0000 0000 0000 0000 0000    ................
mdh >>x
0x00001080   0000 0000 0000 0000 0000 0000 0000 0000    ................
Mbug >>
```

mdb mdb

Display Memory Byte

mdb address
mdb start +
mdb start - end

This command will display memory as 16 bytes per line of output. If the “range” form is used and it exceeds 16 words then memory display will enter interactive mode. In interactive mode, the user needs to type <return> to continue or “x” to exit. The start address is normalized to the previous quad-word boundary. Likewise, the ending address is normalized to the next quad-word boundary. For example, if the start address was 0x00000104 then the first memory address to be displayed would be 0x00000100. If the end address was 0x00000104 then the last location to be displayed would be 0x0000010C.

The “plus” form will cause the command to enter interactive mode and display memory in 16-word blocks from a given start address and continue until the user types “x”.

Note that the above parameter forms can be combined by separating the forms with a comma or whitespace. This will display multiple portions of the memory space with one command.

Examples:

```
Mbug >>mdb 1000,3000
0x00001000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
Mbug >>mdb 2000-2040
0x00002000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00002010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00002020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00002030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00002040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
Mbug >>mdb 3000+
0x00003000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
mdb >>
0x00003040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
mdb >>x
0x00003080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x000030a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x000030b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

```
Mbug >>
```

menu me

Show Mbug Command List

menu

me

?

This command will list all of the commands that are available in the current implementation of Mbug.

Examples:

```
Mbug >> menu
COMMAND LIST
Command                  Mnemonic      Command                  Mnemonic
=======                  ========      =======                  ========
About...                 about         Memory Fill              mf
Assemble                 as            Memory Modify            mm
Breakpoint Set           br            Memory Modify Halfword   mmh
Breakpoint List          br            Memory Modify Byte       mmb
Breakpoint Delete        nobr          Memory Move              mv
Define Alias             da            Memory Search            ms
Disassemble              ds            Register Display         rd
Download                 dl            Register Modify          rm
Go                       go            Run Alias                ra
Help                     help          Set Input                si
Log session              log           Transparent Mode         tm
Menu                     menu or '?'   Trace                    tr
Memory Display           md            Verify download          ver
Memory Display Halfword  mdh
Memory Display Byte      mdb           Repeat last command      '.'

For additional details, type 'help <command>'
Mbug >>
```

mf

Memory Fill

mf start end data

The range of memory spanning from the starting address to the ending address is filled in with the given 32-bit data pattern. The fill is inclusive of the end point. Address arguments must be word aligned.

Examples:

```
Mbug >> mf 50100 50200 89898989
Mbug >> mf 50140 5015c 00000000
Mbug >> md 50120-50160
0x00050120  89898989 89898989 89898989 89898989         ................
0x00050130  89898989 89898989 89898989 89898989         ................
0x00050140  00000000 00000000 00000000 00000000         ................
0x00050150  00000000 00000000 00000000 00000000         ................
0x00050160  89898989 89898989 89898989 89898989         ................
Mbug >> mf 50144 50144 44444444
Mbug >> md 50120-50160
0x00050120  89898989 89898989 89898989 89898989         ................
0x00050130  89898989 89898989 89898989 89898989         ................
0x00050140  00000000 44444444 00000000 00000000         ....DDDD........
0x00050150  00000000 00000000 00000000 00000000         ................
0x00050160  89898989 89898989 89898989 89898989         ................
Mbug >>
```

mm

Memory Modify Word

mm address
mm start +
mm start - end

Memory modify is an interactive command. It will display the contents of the given memory address and allow the user to change the value stored there. Memory is considered to be a contiguous set of 32-bit integers. Addresses must be word aligned.

The “plus” form causes the command to start at a given address and continue until the end of memory or until the user types “x” to exit the memory modify loop.

The “range” form allows modifications for the inclusive range from start to end. When the end address is reached the memory modify loop is automatically exited. The user can type “x” at any time to exit the memory modify loop.

Examples:

```
Mbug >> memod 50100

     0x00050100 : 0x89898989 : ?  44444444

Mbug >> memod 50110-50118

     0x00050110 : 0x89898989 : ?  11111111
     0x00050114 : 0x89898989 : ?  22222222
     0x00050118 : 0x89898989 : ?  33333333

Mbug >> memod 50200+

     0x00050200 : 0x89898989 : ?  12341234
     0x00050204 : 0x00000000 : ?  12341234
     0x00050208 : 0x00000000 : ?  x

Mbug >> memod 50240-50270

     0x00050240 : 0x00000000 : ?  2133a234
     0x00050244 : 0x00000000 : ?  ffdfaddf
     0x00050248 : 0x00000000 : ?  x

Mbug >>
```

mmh

mmh

Memory Modify Halfword

mmh address
mmh start +
mmh start - end

Memory modify halfword is an interactive command. It will display the contents of the given memory address and allow the user to change the value stored there. Memory is considered to be a contiguous set of 16-bit integers. Addresses must be half-word aligned.

The “plus” form causes the command to start at a given address and continue until the end of memory or until the user types “x” to exit the memory modify loop.

The “range” form allows modifications for the inclusive range from start to end. When the end address is reached the memory modify loop is automatically exited. The user can type “x” at any time to exit the memory modify loop.

Examples:

```
Mbug >>mmh 3400

     0x00003400 : 0x0000 : >>  1234

Mbug >>mmh 3400-3406

     0x00003400 : 0x1234 : >>  8989
     0x00003402 : 0x0000 : >>  aaaa
     0x00003404 : 0x0000 : >>  5555
     0x00003406 : 0x0000 : >>  4321

Mbug >>mmh 3400+

     0x00003400 : 0x8989 : >>  3333
     0x00003402 : 0xaaaa : >>  1111
     0x00003404 : 0x5555 : >>  6666
     0x00003406 : 0x4321 : >>  8888
     0x00003408 : 0x0000 : >>  1234
     0x0000340a : 0x0000 : >>  x

Mbug >>
```

mmb mmb

Memory Modify Byte

mmb address
mmb start +
mmb start - end

Memory modify byte is an interactive command. It will display the contents of the given memory address and allow the user to change the value stored there. Memory is considered to be a contiguous set of 8-bit integers. There are no alignment restrictions on the addresses.

The “plus” form causes the command to start at a given address and continue until the end of memory or until the user types “x” to exit the memory modify loop.

The “range” form allows modifications for the inclusive range from start to end. When the end address is reached the memory modify loop is automatically exited. The user can type “x” at any time to exit the memory modify loop.

Examples:

```
Mbug >>mmb 3400

     0x00003400 : 0x33 : >>  12

Mbug >>mmb 3400-3405

     0x00003400 : 0x12 : >>  55
     0x00003401 : 0x33 : >>  aa
     0x00003402 : 0x11 : >>  89
     0x00003403 : 0x11 : >>  fa
     0x00003404 : 0x66 : >>  23
     0x00003405 : 0x66 : >>  65

Mbug >>mmb 3400+

     0x00003400 : 0x55 : >>  ee
     0x00003401 : 0xaa : >>  45
     0x00003402 : 0x89 : >>  32
     0x00003403 : 0xfa : >>  x

Mbug >>
```

ms ms

Memory Search

ms **start end data**

This command will search the block of memory from start to end for the specified (word-size) data. All matches within the range will be reported. If no match is found, nothing is printed. Leading zeros are automatically added to determine the full search word. Both start and end addresses must be word aligned.

Examples:

```
Mbug >> md 50120-50160
0x00050120  44000044 00000000 00000000 00000000         D..D............
0x00050130  00000000 00000000 00000000 00000000         ................
0x00050140  00004400 00000000 12340000 00000000         ..D......4......
0x00050150  00000000 00001234 12340000 00000044         .......4.4.....D
0x00050160  00000000 00000000 00000000 00000000         ................
Mbug >> ms 50120 50160 12340000
        0x00050148
        0x00050158
Mbug >> ms 50120 50160 44
        0x0005015c
Mbug >> ms 50120 50160 1234
        0x00050154
Mbug >>
```

mv

memory move

mv **start end dest**

This command will move the block of memory between start and end to a destination block of memory of the same size starting at dest. All three addresses must be word aligned. Data is transferred as 32-bit quantities.

Examples:

```
Mbug >> mf 50100 50110 ffffffff
Mbug >> md 50100-50150
0x00050100      ffffffff ffffffff ffffffff ffffffff        ................
0x00050110      ffffffff 00000000 00000000 00000000        ................
0x00050120      00000000 00000000 00000000 00000000        ................
0x00050130      00000000 00000000 00000000 00000000        ................
0x00050140      00000000 00000000 00000000 00000000        ................
0x00050150      00000000 00000000 00000000 00000000        ................
Mbug >> mv 50100 50110 50140
Mbug >> md 50100-50150
0x00050100      ffffffff ffffffff ffffffff ffffffff        ................
0x00050110      ffffffff 00000000 00000000 00000000        ................
0x00050120      00000000 00000000 00000000 00000000        ................
0x00050130      00000000 00000000 00000000 00000000        ................
0x00050140      ffffffff ffffffff ffffffff ffffffff        ................
0x00050150      ffffffff 00000000 00000000 00000000        ................
Mbug >>
```

nobr

No Breakpoint

nobr **address**

nobr

The No breakpoint command (**nobr**) will delete a breakpoint that was previously set at a specified address in memory. This command will remove one breakpoint at a time. If several breakpoints (but not all) are to be deleted, this command must be executed once for each address. Address arguments must be halfword aligned.

All currently set breakpoints are removed by invoking **nobr** without the address argument.

Examples:

```
Mbug >>br 3400
breakpoint set at 0x00003400
Mbug >>br 3420
breakpoint set at 0x00003420
Mbug >>br 3440
breakpoint set at 0x00003440
Mbug >>br
Current Breakpoint List:
      0x00003400
      0x00003420
      0x00003440
Mbug >>nobr 3420
breakpoint deleted from 0x00003420
Mbug >>br
Current Breakpoint List:
      0x00003400
      0x00003440
Mbug >>nobr
All breakpoints removed.
Mbug >>br
Current Breakpoint List:
Mbug >>
```

rd rd

Register Display

rd	**r**	- entire general register family
rd	**rx**	- one general purpose register
rd	**rx+**	- from rx to r15
rd	**rx-ry**	- from rx to ry
rd	**a**	- entire alternate general register family
rd	**ax**	- one alternate general register
rd	**ax+**	- from ax to a15
rd	**ax-ay**	- from ax to ay
rd	**crx**	- one control register

This command will display the contents of the specified registers, while providing the user with several options for specifying the registers to be displayed. The whole family of general purpose registers or alternate general registers can be viewed by typing "**rd r**" or "**rd a**" respectively. A single register can be viewed by specifying rx, ax, or crx, where the first character(s) denotes the register family and the *x* denotes the register number. Control registers may be selected by their standard abbreviations as well as their register number.

The "plus" form displays the contents of the register family starting with the given register up to and including the last register in that family.

The "range" form displays the contents of the registers from *x* to *y*.

Note that the "entire family", "plus", and "range" forms are not available in the control register family.

The above parameter forms can be combined by separating them with a comma or whitespace. This will display multiple registers in different register families with one command. Note that the general and alternate register displays are aligned on an even-numbered register boundary, so if an even numbered register needs to be displayed, the odd-numbered register following it is also displayed.

Examples:

```
Mbug >>rd r4+
r04:  0x00000000        r05:  0x00000000
r06:  0x00000000        r07:  0x00000000
r08:  0x00000000        r09:  0x00000000
r10:  0x00000000        r11:  0x00000000
r12:  0x00000000        r13:  0x00000000
r14:  0x00000000        r15:  0x00000000
```

```
Mbug >>rd psr
psr : 0x00000000
Mbug >>rd r a4-a7 cr3 ss0
r00:  0x00000000        r01:  0x0000ff00
r02:  0x00000000        r03:  0x00000000
r04:  0x00000000        r05:  0x00000000
r06:  0x00000000        r07:  0x00000000
r08:  0x00000000        r09:  0x00000000
r10:  0x00000000        r11:  0x00000000
r12:  0x00000000        r13:  0x00000000
r14:  0x00000000        r15:  0x00000000

a04:  0x00000000        a05:  0x00000000
a06:  0x00000000        a07:  0x00000000

fpsr : 0x00000000
ss0 : 0x00000000
Mbug >>
```

rm

Register Modify

rm	**r**	- entire general register family
rm	**rx**	- one general purpose register
rmt	**rx+**	- from rx to r15
rm	**rx-ry**	- from rx to ry
rm	**a**	- entire alternate general register family
rm	**ax**	- one alternate general register
rm	**ax+**	- from ax to a15
rm	**ax-ay**	- from ax to ay
rm	**crx**	- one control register

This will display the contents of the specified registers and allow the user to modify them. There are several input formats for viewing and modifying the registers. To view and modify the entire family of general purpose registers or alternate general registers, the user can type “**rm r**” or “**rm a**” respectively. When this format is used, all of the registers of the respective family will be displayed on the screen one at a time beginning with register 0 and ending when register 15 is reached or when the user types “x”.

A single register can be viewed and modified by specifying rx, or ax, or crx, where the first character(s) denotes the register family and the *x* denotes the register number. Control registers may be selected by their standard abbreviations as well as their register number.

The “plus” form is used to display the contents of the register starting with the given register and ending when register 15 is reached or when the user types “x”.

The “range” form displays the contents of the registers from *x* to *y*.

Note that the “entire family”, “plus”, and “range” forms are not available in the control register family.

Note that the above parameter forms can be combined by separating them with a comma or whitespace. This will modify registers in different register families with one command.

Starting in Version 1.5, these command mnemnic has been shortened from **rmm** and **regmodmult** to **rm**.

Example:

```
Mbug >>rm r
     r00 = 0x00000000 : rm >>12345678
     r01 = 0x0000ff00 : rm >>87654321
     r02 = 0x00000000 : rm >>1
     r03 = 0x00000000 : rm >>x

Mbug >>rm r2-r4
     r02 = 0x00000001 : rm >>aaaa5555
     r03 = 0x00000000 : rm >>5555aaaa
     r04 = 0x00000000 : rm >>99999999

Mbug >>rm r14+
     r14 = 0x00000000 : rm >>ffff00ff
     r15 = 0x00000000 : rm >>11001111

Mbug >>rm cr2, a5
epsr : 0x00000000

 new value ? rmm >>80008000

     a05 = 0x00000000 : rm >>fefefefe

Mbug >>rm a15
     a15 = 0x00000000 : rm >>12345678

Mbug >>rm r a cr2
r00:  0x12345678        r01:  0x87654321
r02:  0xaaaa5555        r03:  0x5555aaaa
r04:  0x99999999        r05:  0x00000000
r06:  0x00000000        r07:  0x00000000
r08:  0x00000000        r09:  0x00000000
r10:  0x00000000        r11:  0x00000000
r12:  0x00000000        r13:  0x00000000
r14:  0xffff00ff        r15:  0x11001111

a00:  0x00000000        a01:  0x0000ff00
a02:  0x00000000        a03:  0x00000000
a04:  0x00000000        a05:  0xfefefefe
a06:  0x00000000        a07:  0x00000000
a08:  0x00000000        a09:  0x00000000
a10:  0x00000000        a11:  0x00000000
a12:  0x00000000        a13:  0x00000000
a14:  0x00000000        a15:  0x12345678

epsr : 0x80008000
Mbug >>
```

ra

Run Alias

ra

This instruction will read in the string which the user has defined as an alias. Then, the commands in this string will be executed sequentially.

Example:

Mbug >> **ra**

The runalias command can also be embedded within a command line. For example, if the alias string has previously been defined as

tr +; **rd r**

Typing the command

Mbug >> **log**; **trace 2100; ra**; **log**

is identical to typing

Mbug >> **log**; **trace 2100; tr** +; **rd r**; **log**

See Define Alias (**da**) for a complete example on defining the alias.

si

Set serial I/O Configuration

si both
si key
si ?

This is used to tell the firmware (Mbug) which port to use as the serial i/o.

key All i/o (terminal interaction as well as downloads) will go through the keyboard serial port. This allows the user to connect an evaluation board equipped with Mbug to a single serial line (for example a PC or Unix Workstation) without losing any functionality.

both Terminal interaction will go through the keyboard serial port while host interaction (downloads) will go through the host serial port.

? This will return one of the arguments described above, indicating the current setting.

The default is **key**.

The log and transparent mode functions are automatically disabled if Set Input has been used to indicate that all I/O will go through the keyboard serial port.

Example:

```
Mbug >> si ?
Set Input Port : Both Ports Active
Mbug >> si key
Set Input Port : set to Keyboard Port
Mbug >> si both
Set Input Port : set to Both Ports
Mbug >>
```

tr tr

Single Step Trace

tr **address**

tr +

This command allows the user to single-step through a user program. The microprocessor will execute a single instruction, and then return control back to the firmware. If a specific address is given, then a single instruction is executed from that address. However, if the "plus" form is used, then the address of the instruction to execute is derived from the EPC (Exception Program Counter) register. After the instruction has been executed, control is returned to the firmware (Mbug) and the user can examine the programming model or continue to trace through instructions.

An rte instruction is used to transfer control from the monitor to user code. The Trace command copies the start address into the EPC (Exception Program Counter) register (if supplied). The rte instruction then loads the PC from the EPC and the PSR from the EPSR (Exception Processor Status Register). The user must have previously written EPSR to the desired value.

The "plus" form will continue execution at the address already contained in the EPC register. This is useful for tracing through a sequence of instructions.

Example:

```
Mbug >> disassem 2100
0x00000000 0x1011 mfcr    r1,cr1
Mbug >> trace 2100
A Run Mode or Trace exception has occurred.
Current Instruction Pointer: 0x00002102 0x0064 ldm     r4-r15,(r0)
Mbug >> trace +
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002104 0xbeef st.b    r15,(r14,0xe)
Mbug >> .
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002106 0x1021 mfcr    r1,cr2
Mbug >>
```

Enter Transparent Mode

tm

This command will put Mbug into a transparent mode. In other words, as the user types data into the keyboard, the data is sent directly to the host serial port. In addition, data that comes in from the host serial port will be forwarded to the keyboard serial port. This allows the target board to speak directly to the host system. Very often, the host machine is a Unix system and using this command will provide access to a Unix operating system for use along with the target board. This command is also used for retrieving downloadable files. The user can break out of transparent mode by typing <ctrl>-a. (To use this feature, the Set Input setting must be both).

Example:

```
Mbug >> tm
```

(On the host machine (Unix):)

```
Unix $ ls
hello.c
hello.o
hello.srec
Unix $ sleep 1; cat hello.srec(<ctrl>-a)
```

(On the target board running Mbug:)

```
Mbug >> dl
Download Complete.
Mbug >>
```

ver

Verify Data From The Host

ver

This command provides the ability to verify data received from the host serial port. The data can be received in S-record format. The data is compared on a byte by byte basis against memory locations specified by the input file. The first mismatch found will terminate the command with an error report providing the address, expected data and actual data.

Verify calculates and validates the checksum information provided in the downloaded file. Any mismatch terminates the command and the error is reported to the terminal.

Example :

```
Mbug >> tm
```

(On the host machine (Unix):)

```
Unix $ ls
hello.c
hello.o
hello.srec
a.out
Unix $ cat hello.srec (<ctrl>-a)
```

(On the target board running Mbug:)

```
Mbug >> ver
Verify Complete.
Mbug >>
```

Example :

```
Mbug >> ver
```

(On the host machine (PC):)

Use the terminal emulator’s pull-down menu’s to initiate transfer of the S-record file.Transfer mode should be text or ASCII for S-Records.

(On the target board running Mbug:)

```
Verify Complete.
Mbug >>
```

CHAPTER 4 : SINGLE LINE ASSEMBLER/ DISASSEMBLER

4.1 Assembler Syntax

The single line assembler and disassembler follow the M.CORE Application Binary Interface (ABI) standard wherever possible. There are some exceptions, however due to the nature of single line assembly versus file based assembly. The following subsections detail the

4.1.1 Labels

Labels are not allowed by the single line assembler and will be flagged as a syntax error.

4.1.2 Register Specifiers

General purpose registers are specified as rn where n is an integer in the range of 0 through 15 inclusive. No distinction is made between the general and alternate registers files.

Control Registers are specified as crn where n is a number from 0 through 31 inclusive. Unimplemented control registers are not flagged by the assembler. The single line assembler does not currently support the common names assigned to the control registers (eg., PSR is not recognized as an alias to cr0).

4.1.3 Immediate Constants

Immediate constants are assumed to be decimal (Base 10) unless preceded by “0x” indicating that they are hexadecimal (Base 16). Octal and character constants are not supported. Offset and modulo32 immediates (zero indicates a value of 32) follow the syntax specified in the M.CORE Programmer’s Reference Manual. The assembler will adjust the immediate as appropriate before inserting it into the opcode.

Examples:

```
Mbug >>as 4000+
0x00004000 0x0000 bkpt          >> movi r4,45
0x00004002 0x0000 bkpt          >> movi r4,0x45
0x00004004 0x0000 bkpt          >> addi r3,32
0x00004006 0x0000 bkpt          >> bmaski r5,32
0x00004008 0x0000 bkpt          >> bmaski r5,0
ERROR : operand out of range
0x00004008 0x0000 bkpt          >> x
Mbug >>ds 4000-4008
0x00004000 0x62d4 movi    r4,0x2d
0x00004002 0x6454 movi    r4,0x45
0x00004004 0x21f3 addi    r3,0x20
0x00004006 0x2c05 bmaski  r5,0x20
0x00004008 0x0000 bkpt
Mbug >>
```

4.1.4 Load/Store Addressing

Load and Store instruction indirect address specifiers follow the syntax documented in the M.CORE Programmer’s Reference Manual. The displacement values are specified in bytes;

however, for the halfword and word sized operations a misaligned offset will be truncated to the next lower aligned boundary. This operation is consistent with the scaled offset value stored in the opcode. The parenthesis surrounding the address specifier are required.

The default load/store size is word. Size modifiers are used to specify anything other than the default. “.b” (or, simply “b”) specifies a byte sized operation; “.h” or “h” a halfword sized operation; and, “.w” or “w” a word sized operation.

Example:

```
Mbug >>as 5000+
0x00005000 0x0000 bkpt >> ld.b r4,(r2,1)
0x00005002 0x0000 bkpt >> ldb r4,(r2,2)
0x00005004 0x0000 bkpt >> ld.h r4,(r2,1)
0x00005006 0x0000 bkpt >> ldh r4,(r2,2)
0x00005008 0x0000 bkpt >> ld.w r4,(r2,1)
0x0000500a 0x0000 bkpt >> ldw r4,(r2,2)
0x0000500c 0x0000 bkpt >> ld r4,(r2,4)
0x0000500e 0x0000 bkpt >> x
Mbug >>ds 5000-500e
0x00005000 0xa412 ld.b r4,(r2,0x1)
0x00005002 0xa422 ld.b r4,(r2,0x2)
0x00005004 0xc402 ld.h r4,(r2,0x0)
0x00005006 0xc412 ld.h r4,(r2,0x2)
0x00005008 0x8402 ld r4,(r2,0x0)
0x0000500a 0x8402 ld r4,(r2,0x0)
0x0000500c 0x8412 ld r4,(r2,0x4)
0x0000500e 0x0000 bkpt
Mbug >>
```

4.1.5 Branch Offsets

Branch offsets are one of the places where the single line assembler differs from the ABI. Branch offsets are to be specified as the actual target address. The assembler will calculate the offset based on the current memory address pointer and use the calculated value to construct the opcode.

Branch offsets default to decimal, unless overridden by the “0x”modifier to specify a hexadecimal value.

Example:

```
Mbug >>as 5000
0x00005000 0x0000 bkpt >> br 0x5000
Mbug >>ds 5000
0x00005000 0xf7ff br 0x00005000
Mbug >>
```

4.1.6 LRW Offsets

LRW offsets follow the same convention as branch offsets; the offset is entered as the actual target address and the assembler calculates the correct offset to put in the opcode. The value must be surrounded by square brackets (“[<value>]”) as documented in the M.CORE Programmer’s Reference Manual.

Example:

```
Mbug >>as 4000+
0x00004000 0x0000 bkpt >> lrw r6,[0x4000]
0x00004002 0x0000 bkpt >> br 0x4008
0x00004004 0x0000 bkpt >> x
Mbug >>mm 4004
0x00004004 : 0x00000000 : >> 12345678
Mbug >>ds 4000-4008
0x00004000 0x7600 lrw r6,[0x00004000]
0x00004002 0xf002 br 0x00004008
0x00004004 0x1234 mov r4,r3
0x00004006 0x5678 DATA 0x5678
0x00004008 0x0000 bkpt
Mbug >>
```

CHAPTER 5 : Mbug EXCEPTION SUPPORT

There are 17 exception traps in this version of Mbug. A message indicating which exception has occurred is displayed for all of them except System Reset. Table 3: *Exceptions Trapped and Identifed by Mbug* lists the exceptions trapped by Mbug Version 1.0. All vector addresses not specifically listed vector through a common reserved vector routine.

Table 3: Exceptions Trapped and Identifed by Mbug

Vector Offset	Exception Type
0x0000	System Reset
0x0004	Misaligned Access
0x0008	Access Error
0x000C	Divide by Zero
0x0010	Illegal Instruction
0x0014	Pivilege Violation
0x0018	Trace Exception
0x001C	Breakpoint Exception
0x0020	Unrecoverable Error
0x0024	Soft Reset
0x0028	Normal Interrupt Autovector
0x002C	Fast Interrupt Autovector
0x0030	Hardware Accelerator
0x0040	Trap 0 Instruction
0x0044	Trap 1 Instruction
0x0048	Trap 2 Instruction
0x004C	Trap 3 Instruction

System Reset occurs when the software is booted up or the evaluation board is reset. The other exceptions occur due to interrupts or errors in the execution of the code. For details on what causes each exception, see the M.CORE Programmer's Reference Manual.

For interrupt-driven I/O, an exception will also occur when <ctrl>-c is invoked by the

user. When <ctrl>-c is pressed, Mbug aborts the user code and gives control back to the firmware. The MMC2001 version of Mbug utilizes polled I/O, therefore the <ctrl>-c interrupt feature is not available.

The user is notified of an exception by a message that appears on the terminal. Control is returned to the firmware. If the exception was caused by the completion of a trace or by arriving at a breakpoint during execution of the user's code, the user can continue testing. Otherwise the user may need to modify the code to correct a problem and download the program again to resume testing.

CHAPTER 6 : RESTRICTIONS

6.1 Mbug use of Control Registers

There are five Supervisor Scratch (ss0-ss4) registers, numbered 0 through 4. Mbug makes use of ss3 and ss4, so any user values placed into these two registers will be destroyed whenever control is returned to the monitor. The user is encouraged to place any values that are of interest or necessity into only ss0 through ss2, although the user can use the other two scratch registers for calculations or temporary storage.

6.2 Interrupt Driven I/O

Interrupt driven I/O is not active in Version 1.0 of Mbug. Some code has been written for this functionality, but it has not been tested. Therefore, interruption of execution using <ctrl>-c is not supported.

6.3 Memory Test

No diagnostic memory tests have been included in Mbug. The user must incorporate tests that will provide this function, if desired.

Appendix A : Mbug Command Summary

Command Mnemonic	Command Summary
.	Repeat last command
as	Assemble instruction at address(es)
nobr	Delete breakpoint at address
br	List breakpoints
br	Set breakpoint at address
da	Define alias for command sequence
ds	Disassemble an instruction at address(es)
dl	Download S-Record file to memory
go	Execute from address
he	Show more information on command
log	Record debug session to host
md	Display memory as words
mdh	Display memory as halfwords
mdb	Display Memory as bytes
mf	Fill memory block with data pattern
mm	Modify word at memory address(es)
mmh	Modify halfword at memory address(es)
mmb	Modify byte at memory address(es)
mv	Move memory block to destination
ms	Search memory block for data word
me	Show list of available commands
rd	Display selected register(s)
rm	Modify selected register(s)
ra	Execute commands in the alias
si	Set I/O port
tr	Trace instructions from given address
tm	Enter transparent mode to host
ver	Verify memory with S-records

Appendix B : Mbug Auxillary Commands

Mbug provides an auxillary set of non-interactive commands for use by debuggers and other programs which must communicate with the debugger in a predictable fashion. These commands are summarized in the following table.

Command Mnemonic	Command Summary
as1	Assemble a Single Instruction
mr	Memory Read (Word)
mrh	Memory Read (Halfword)
mrb	Memory Read (Byte)
mw	Memory Write (Word)
mwh	Memory Write (Halfword)
mwb	Memory Write (Byte)
rw	Register Write

Appendix C : Mbug ERROR CODES AND MESSAGES

Parser Errors

0xfb00	UNKNOWN_COMMAND	unrecognized command
0xfb01	UNKNOWN_REGISTER	unknown register
0xfb02	UNKNOWN_CTL_REGISTER	unknown control register
0xfb03	ILLEGAL_RD_STAGE	cannot specify whole register family in range
0xfb04	ILLEGAL_REG_FAMILY	cannot specify a range of control registers
0xfb05	RANGE_CROSS_FAMILY	cannot specify a range across register families
0xfb06	UNIMPLEMENTED_STAGE	invalid rd or rmm parameter format
0xfb07	UNKNOWN_OPERATOR	unknown operator in arguments
0xfb08	INVALID_FILENAME	invalid download filename

Errors from Error Checking Toolbox

0xfd00	INVALID	NOT valid
0xfd01	VALID	valid
0xfd02	INVALID_SIZE	invalid address size
0xfd03	OUT_OF_BOUNDS_ADDRESS	address not in bounds
0xfd04	INVALID_HEX_INPUT	invalid hex address
0xfd05	INVALID_REGISTER	invalid register number
0xfd07	NOT_WORD_ALIGNED	not a word aligned address
0xfd06	NOT_HWORD_ALIGNED	not a half-word aligned address
0xfd08	REVERSED_ADDRESS	start address is greater than end
0xfd09	RANGE_OVERLAP	destination overlaps source addresses
0xfd0a	ERROR	an unidentified error has occured
0xfd0b	INVALID_PARAM	invalid input parameter

Get Argument Errors

0xFE00	INVALID_NUMBER_ARGS	invalid number of arguments
0xFE01	UNKNOWN_PARAMETER	unknown type of command parameter

Tokenizer Toolbox Errors

0xFF00	ILLEGAL_CHARACTER	unrecognized character in input stream
0xFF01	TTL_NOT_SORTED	token translation list not sorted
0xFF02	TTL_NOT_DEFINED	token translation list not assigned
0xFF03	INVALID_STRING	unable to extract string from input stream
0xFF04	BUFFER_EMPTY	input buffer is empty
0xFF05	INVALID_MODE	input buffer is in an unrecognized mode
0xFF06	TOK_INTERNAL_ERROR	internal tokenizer error
0xFF07	TOO_MANY_IBS	too many open input buffers
0xFF08	NO_OPEN_IBS	no open input buffers

0xf400	INSTRUCTION_SYNTAX	instruction syntax incorrect
0xf401	OUT_OF_RANGE	operand out of range
0xf402	ILLEGAL_SEPARATOR	illegal separator or number of arguments
0xf403	ILLEGAL_REG_OPERAND	illegal register operand

Screen Toolbox Errors

0xfc00	RESERVED_WORD	used a reserved word as an argument
0xf404	NUMBER_CONVERSION	Number converison error

Breakpoint Errors

0xFA00	FULL_BPDS	breakpoint data structure is full

Download Errors

0xf900	NOT_IN_S_RECORD_FORMAT	not in S-Record Format
0xf901	UNREC_RECORD_TYPE	unrecognized record type
0xf902	CONVERSION_ERROR	ascii to int conversion error
0xf903	INVALID_MEMORY	bad s-record memory address

Compression and Decompression Errors

0xf800	COMP_UNK_CHARACTER	unknown compressed character
0xf801	COMP_UNKNOWN_STATE	unknown binary state
0xf802	NOT_IN_COMPRESSED_FORMAT	not in compressed S-Record format
0xf904	COMPARE_FAILED	S-record verify with memory failed

Serial I/O Handling Errors

0xf700	UNKNOWN_PORT_STATE	unrecognized serial port configuration
0xf500	TM_NEEDS_BOTH_PORTS	transparent mode needs two serial ports

Register Errors

0xf600	SPR_NOT_FOUND	cannot find in special register file

Appendix D : CMB1200 Evaluation System

D.1 Memory Map

The memory model for Mbug as configured for the CMB1200 Evaluation System is shown in Figure 2-1. Mbug Memory Model (MMC2001Internal ROM). The exception vectors are located within the address range of 0x0000 - 0x01ff. The Mbug code begins at address 0x200 and extends approximately to 0xffff. Initialized data variables are also stored in this ROM address range and copied to RAM during initialization. On-chip RAM runs from 0x3000000 through 0x30007fff. RAM addresses 0x3000000 through 0x300004fff (20k bytes) are reserved for the user, while 0x30005000 - 0x30007fff are used by the monitor. The monitor stack grows down from 0x30007ffc.

D.2 Reset Initialization

Mbug performs a complete initialization of the system in response to a reset from any of the reset sources. The following sections detail the configuration of the system after Mbug initialization.

D.2.1 Reset Source Indication

The source(s) of the hardware reset will be displayed as part of the banner written at the completion of the initialization sequence. The reset source is determined by reading the contents of the Reset Status Register.

Warning: The reset status bits are cleared by a write to the status register. The intialization sequence must write this register to setup the clock output; therefore, user code intiated by Mbug cannot read the reset source register and expect valid data.

D.2.2 Clocks

The clock output is enabled by Mbug with the source derived from the high-ref. clock. Other options can be selected by writing the control re gister through the memory modify command or user code.

D.2.3 Chip Selects

The chip select functions are initilized according to the standard memory configuration on the CMB1200 Evaluation System. Flash memory access time was assumed to be 90nS. RAM wait states were calculated for 70 nS access time memories. All memories are assumed to be 16-bits wide. CS3 was configured for a “generic” 8-bit peripheral with timing setup as loose as possible through the chip select options. All wait state values were calculated for an perating frequency of 32 MHZ. Table 5: *CMB1200 Evaluation System Chip Select Initialization* summarizes the options programmed for each of the chip selects.

Table 5: CMB1200 Evaluation System Chip Select Initialization

CHIP SELECT	ADDRESS RANGE	PORT SIZE	WAIT STATES	SUP./ USER	R/W	LATE OE	LATE CS	EDC
CS0 (FLASH)	`2d000000-2dffffff`	16	4	S/U	Read Only	No	No	No
CS1 (RAM)	`2f000000-2fffffff`	16	3	S/U	R/W	No	No	No
CS2 (RAM)	`2e000000-2effffff`	16	3	S/U	R/W	No	No	N
CS3 (PERIPH)	`2c000000-2fffffff`	8 (D0-D7)	15	S/U	R/W	Yes	Yes	Yes

D.2.4 Show Cycles

Mbug enables show cycles with the options to show all cycles at the cost of performance. The low order data bits are brought out. If the user needs to select other options, the control register can be written with a memory modify command, or from user code.

D.2.5 Internal RAM and ROM

Internal RAM and ROM are intialized so that both supervisor and user accesses are allowed.

D.2.6 User Registers

A virtual copy of the User's registers are stored in system memory. These registers are intialized during the reset intitialization sequence as described below. Note that these values are not actually copied into the CPU registers until user code is executed from the Go or Trace commands. A more detailed description of debugger operation and its relationship to the user can be found in Section ???.

The Processor Status Register (PSR) is initialized to supervisor mode, interrupts disabled, and exceptions enabled.

The Vector Base Register (VBR) is initialized to point at the beginning of the boot memory where Mbug's vectors are located. If the user wishes to move the vector table elsewhere, the trace exception (offset 0x18) and breakpoint exception (offset 0x1c) vectors should be copied from the Mbug table into the user's table. Without these two vectors, Mbug's trace and breakpoint capability cannot be utilized. Mbug supplies handlers for all exceptions and reports all exceptions encountered to the screen. Therefore, the user would be well advised to also copy any other vectors for which a

handler is not provided by the user.

The stack pointer (r15) is initialized to point at the the last longword at the top (highest address) in RAM.

All other user registers are reset to zero.

D.2.7 Software Watchdog

The software watchdog is disabled by Mbug during the reset initialization sequence. The user may enable the watchdog during the execution of user code; however, if control is returned to Mbug without first doing a hardware reset the watchdog will interrupt monitor operation with a watchdog reset once the counter expires.

D.2.8 Time of Day

The Time of Day peripheral module is disabled during Mbug initialization.

D.2.9 Periodic Interrupt

The periodic interrupt module is disabled during Mbug initialization.

D.2.10 Interrupt Controller

The 4 UART interrupt sources are enabled through the interrupt controller to the CPUs Normal interrupt input, although the interrupts are not currently turned on back in the UART. All other interrupt sources are blocked by the interrupt controller.

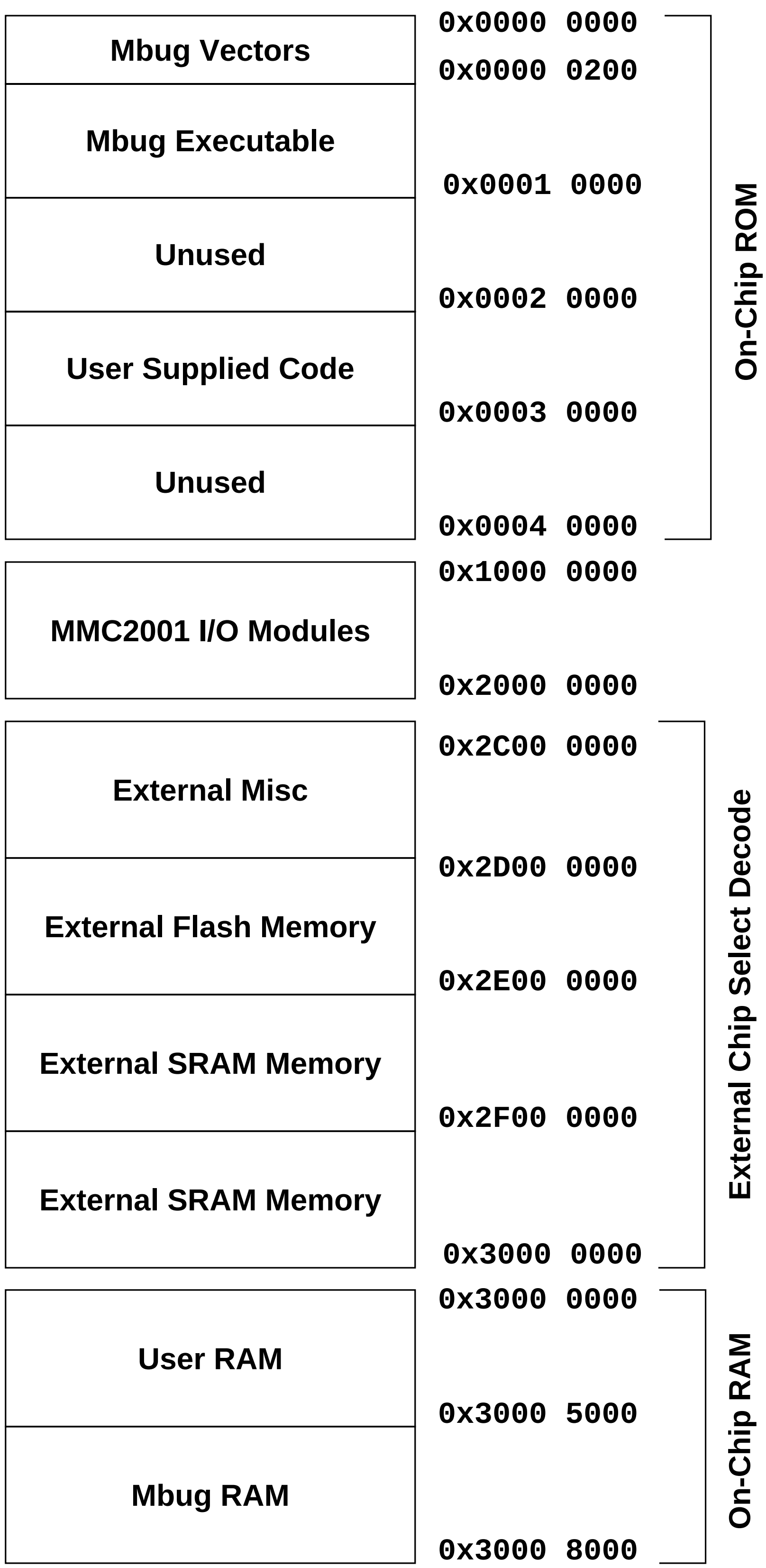
D.2.11 Serial Ports

Both UARTs are initialized by Mbug. UART 0 (Base address 0x10009000) is configured as the Mbug Keyboard port for 19200 baud at 32 MHz high-ref clock input (9600 at 16/16.367 MHz), 8 data bits, 2 stop bits, Rx and Tx interrupts disabled, and hardware handshaking enabled. UART1 (Base address 0x1000a000) is similarily configured with the exception of hardware handshaking and is used as the Host port by Mbug.

D.2.12 Other Peripherals

The remaining MMC2001 peripherals are not configured during Mbug initialization.

Figure 2-1. Mbug Memory Model (MMC2001Internal ROM)



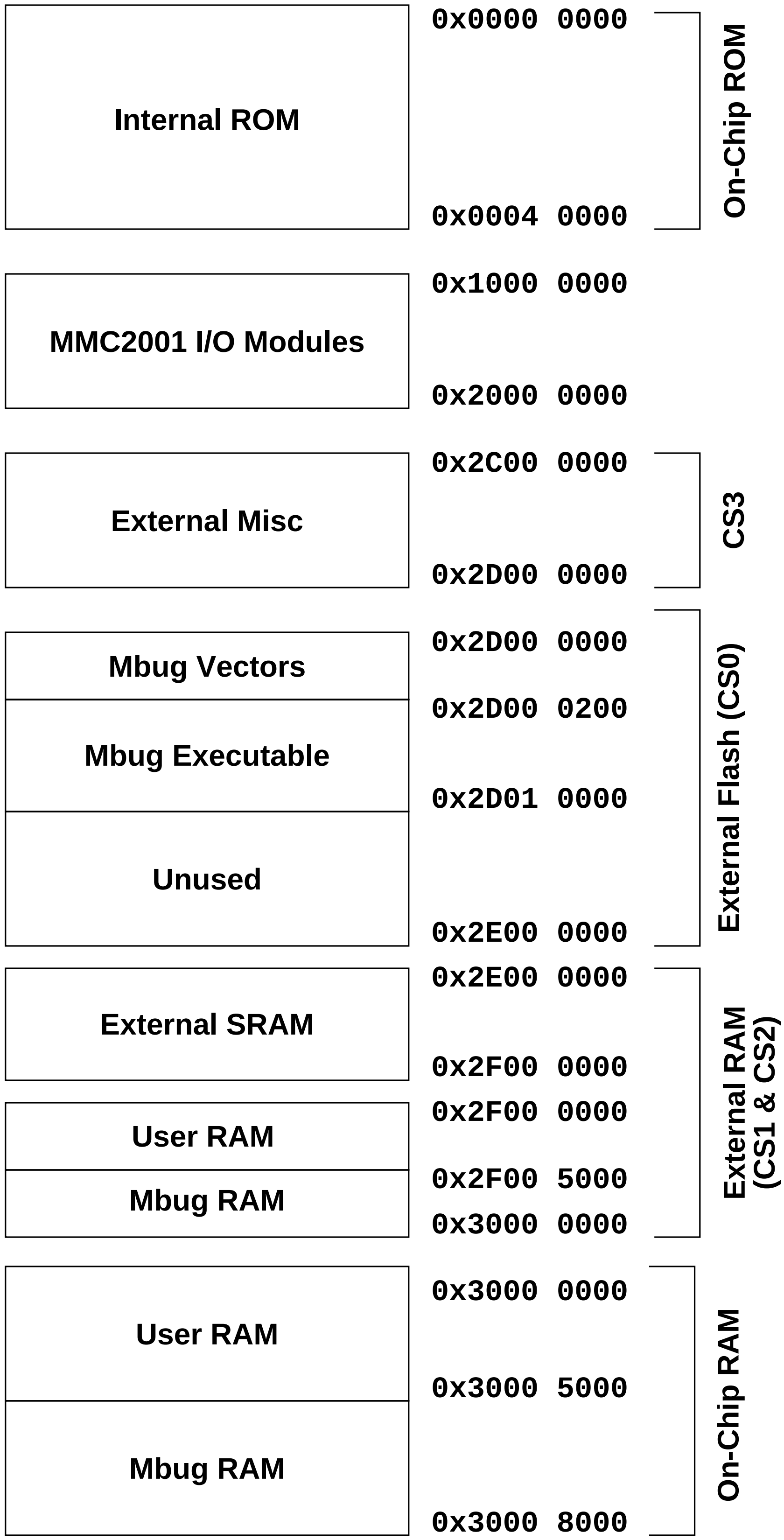


Figure 2-2Mbug Memory Model (CMB1200 Evaluation System)

Appendix E : : ASCII 7-Bit Character Set

<table>
<tr><th>MS
LS</th><th>0
000</th><th>1
001</th><th>2
010</th><th>3
011</th><th>4
100</th><th>5
101</th><th>6
110</th><th>7
111</th></tr>
<tr><td>0
0000</td><td>NUL</td><td>DLE</td><td>SP</td><td>0</td><td>@</td><td>P</td><td>‘</td><td>p</td></tr>
<tr><td>1
0001</td><td>SOH</td><td>DC1</td><td>!</td><td>1</td><td>A</td><td>Q</td><td>a</td><td>q</td></tr>
<tr><td>2
0010</td><td>STX</td><td>DC2</td><td>“</td><td>2</td><td>B</td><td>R</td><td>b</td><td>r</td></tr>
<tr><td>3
0011</td><td>ETX</td><td>DC3</td><td>#</td><td>3</td><td>C</td><td>S</td><td>c</td><td>s</td></tr>
<tr><td>4
0100</td><td>EOT</td><td>DC4</td><td>$</td><td>4</td><td>D</td><td>T</td><td>d</td><td>t</td></tr>
<tr><td>5
0101</td><td>ENQ</td><td>NAK</td><td>%</td><td>5</td><td>E</td><td>U</td><td>e</td><td>u</td></tr>
<tr><td>6
0110</td><td>ACK</td><td>SYN</td><td>&</td><td>6</td><td>F</td><td>V</td><td>f</td><td>v</td></tr>
<tr><td>7
0111</td><td>BEL</td><td>ETB</td><td>‘</td><td>7</td><td>G</td><td>W</td><td>g</td><td>w</td></tr>
<tr><td>8
1000</td><td>BS</td><td>CAN</td><td>(</td><td>8</td><td>H</td><td>X</td><td>h</td><td>x</td></tr>
<tr><td>9
1001</td><td>HT</td><td>EM</td><td>)</td><td>9</td><td>I</td><td>Y</td><td>i</td><td>y</td></tr>
<tr><td>A
1010</td><td>LF</td><td>SUB</td><td>*</td><td>:</td><td>J</td><td>Z</td><td>j</td><td>z</td></tr>
<tr><td>B
1011</td><td>VT</td><td>ESC</td><td>+</td><td>;</td><td>K</td><td>[</td><td>k</td><td>{</td></tr>
<tr><td>C
1100</td><td>FF</td><td>FS</td><td>,</td><td><</td><td>L</td><td>\</td><td>l</td><td>|</td></tr>
<tr><td>D
1101</td><td>CR</td><td>GS</td><td>-</td><td>=</td><td>M</td><td>]</td><td>m</td><td>}</td></tr>
<tr><td>E
1110</td><td>SO</td><td>RS</td><td>.</td><td>></td><td>N</td><td>^</td><td>n</td><td>~</td></tr>
<tr><td>F
1111</td><td>SI</td><td>US</td><td>/</td><td>?</td><td>O</td><td>_</td><td>o</td><td>DEL</td></tr>
</table>

Appendix F : Release Notes

Each release of the debugger firmware is documented below. Numbers in parenthesis refer to the number assigned to the anomoly in the bug/eratta database.

F.1 Revision 1.0

This version was released in the internal ROM on the MMC2001 (Powerstrike) Revision 0 silicon. Off-chip hardware configuration is consistent with the CMB1200 Evaluation System.

F.2 Revision 1.1

Internal development version only. Not formally released. This version fixes the problems identified in Version 1.0: PSR updates on Go and Trace commands (1), and assembly of DIVU instructions with a result register other than r1 (2). This version also contains numerous code rewrites for portability across platforms.

F.3 Revision 1.2

Initially released configured for the external flash on the CMB1200 Evaluation System. Initialization of VBR tracks the boot address starting in this version (4). It also corrects the reversed register fields on disassembly of load/store instructions (3).

F.4 Revision 1.3

Release configured for the external flash on the CMB1200 Evaluation System. Rewrote the way initialization of VBR tracks boot address. Changed port initialization to key only.

F.5 Revision 1.3.1

Code was not changed from 1.3, but link location of RAM has been changed to correspond to CS1 address range.

F.6 Revision 1.4

Internal development version only. Not formally released. Added disable of echo during downloads to fix recursion problem with I/O routines (6). Also added code to recover after monitor exceptions by jumping back to main loop (5).

F.7 Revision 1.5

Initially released configured for the external flash on the CMB1200 Evaluation System. Corrects problems with GO and TRACE commands requiring word aligned addresses (7). Adds warm start routine for gracefull recovery from exceptions during monitor operation (8). Adds new breakpoint command names and capability to

remove all breakpoints at once. Changed register modify command name to RM from RMM. All old commands still recognized. Updates help messages to document new commands.

F.8 Revision 1.6

Initially released configured for the external flash on the CMB1200 Evaluation System. Corrects remaining problems with GO and TRACE commands to halfword aligned addresses (9). Also fixes word opcode size problems in non-interactive assembler (as1) command (10). New Memory Display halfword and byte commands added (11). Help reformatted and long form commands deleted to reduce ROM footprint below 64k.

Mbug

Interactive Debugger for M.CORE Embedded Processors

User’s Guide

Applies to Mbug Versions 1.6 and later

CHAPTER 1 : INTRODUCTION

Mbug is a flexible software tool enabling evaluation and debugging of both hardware and software developed for M.CORE(tm) 32-bit embedded processors. The comprehensive command set gives the application developer the ability to examine and modify any software accesible hardware resource, assemble and disassemble programs in the target memory, and download programs into the target memory. Additional commands provide the developer the capability to set breakpoints, run programs, trace instructions, and trap exceptions.

This document describes the Mbug features, command set, utilities, platform dependent characteristics, error messages , exception traps, and restrictions. Appendices document platform dependent characteristics as well as providing direction on reconfiguration and extension of Mbug.

1.1 Mbug Features

The Mbug software package provides:

- Modification and display of general purpose, alternate, and control registers
- Assembly and disassembly of M.CORE instructions for modification and display of code
- Single-step trace and continued execution from a specified address
- Modification, display, and movement of system memory
- Setting, displaying, and removing breakpoints
- Extensive on-line help
- Ability to execute user-assembled and/or downloaded software in a controlled environment
- Automatic decompression of compressed s-record files while downloading
- Logging function for generating a transcript of a debugging session

CHAPTER 2 : Mbug OVERVIEW

The following sections describe the minimum required hardware configuration, and the memory model.

2.1 General Hardware Requirements

This Mbug software package can be executed on systems that include a minimum configuration of:

2.1.1 M.CORE CPU

Mbug Version 1.0 supports the M.CORE 1 CPU including implementation specific registers and exceptions.

2.1.2 64k-byte Program Memory

The executable code, read-only data and initialized data sections are stored in the program memory and require approximately 64k bytes of storage.

Mbug program memory may reside in on-chip non-volatile memory, external non-volatile memory, or external RAM. In each case, however, the code expects to gain control of the system immediately out of reset in order to initialize the system to a known state.

2.1.3 12k-byte RAM

System RAM is used for initialized data, uninitialized data, and serial I/O buffers. Initialized data is copied from ROM to RAM during initialization.

In the standard configuration, RAM requirements total approximately 12k bytes. Of this total, slightly over 8k bytes are allocated to the serial I/O buffers. Less RAM could be allocated, as long as hardware handshaking is enabled on the serial port connected to the host computer for downloads and transparent mode operation. Changing the size of the serial I/O buffers requires a re-compilation of the monitor program.

2.1.4 Serial I/O Ports

Mbug requires a minimum of one serial port for proper operation, although two are required for some applications. Version 1.0 of the software has been configured to use the two serial ports on the MMC2001 chip. With different I/O drivers, any asynchronous serial peripheral could be used. Changing the serial port I/O drivers requires a re-compilation of the monitor program.

2.2 Mbug Memory Map

The Mbug memory map is hardware (silicon and board) dependent. A detailed diagram can be found in the appendices.

CHAPTER 3 : Mbug COMMANDS

This chapter describes the Mbug command line syntax and command functionality.

3.1 Command Line Syntax

Commands are entered with the following syntax. Multiple commands can be entered on the same line if separated by a semicolon, “;”.

```
Mbug >>[<delimiter>]<command><delimiter[<parameters>]<command_terminator>

<delimiter> := {\ | ,| \t}

<command> := valid command name

<parameters> := <parameter> [<delimiter> <parameters>]

<parameter> := valid parameter for <command>

<command_terminator> := {\n | \r | ;}
```

3.2 Command Parameter Number Syntax

All input numbers to the command line are hexadecimal. a leading ”0x” is not required, nor is it allowed. Leading zeros are optional. The hexadecimal input restriction does not apply to the single line assembler. See CHAPTER 4 : SINGLE LINE ASSEMBLER/DISASSEMBLER for a description of the single line assembler syntax.

Some commands support address ranges. Ranges are specified on the command line as two addresses separated by a dash, “-”. Open-ended address ranges are also supported, and are specified as a single address followed by a plus, “+”.

3.3 Command Parameter Register Syntax

The general purpose registers are specified as r*x*, where *x* is an integer from 0 to 15. Alternate registers are specified by a*x*, where *x* is an integer from 0 to 15. Control registers are specified as cr*x*, where *x* is an integer from 0 to 31. Since not all control registers are implemented on all M.CORE implementations, some numbers in the range may not be valid. For instance, M.CORE 1 only implements the first 13 registers so only 0-12 are valid. Control registers may also be referenced by their standard mnemonic.

The single letter “r” can be used as a shorthand notation to reference the entire general purpose register set. Likewise, the letter “a” can be used to reference the entire alternate register set. A shorthand specifier for the control register family is not provided since they are not considered a family in the same sense as the general and alternate registers.

3.4 Terminating Interactive Commands

The single letter “x” will terminate a command that is operating in interactive mode.

3.5 Command Detailed Descriptions

Detailed descriptions are supplied for each of the Mbug commands. These descriptions include the command name, command line parameter options, a detailed description of command operation, and an example(s) of the command usage. User-entered commands appear in boldface throughout this chapter.

. (period)　　　　　　　　　　　　　　　　　.

repeat last command

.

Typing a period will repeat the last command entered.

Example:

```
Mbug >> trace 2100
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002104  st.w r13,(r01,0)
Mbug >> trace +
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002106 addu r13,r01
Mbug >> .
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002108 mtcr r13,ss0
Mbug >>
```

about about

Mbug Version Information

about

The version information for the current implementation of the Mbug monitor will be displayed on the terminal.

Example:

```
Mbug >> about
M     M
MM   MM
M M M M
M  M  M
M     M bug

Version: 1.6

Copyright Motorola Inc., 1993, 1994, 1995, 1997, 1998

Mbug >>
```

as

Single-Line Assembler

as address
as start+
as start - end

The single-line assembler for the Mbug system will display the contents of memory at the given location and enter interactive mode. The user will be queried for a valid mnemonic and operands which will be assembled into a valid opcode and stored at that memory location. A location can be left unmodified by typing <return> to pass over it.

The “plus” form of the command will allow the user to start assembling code at a given start location and continue through increasing addresses until terminated by the user. The “range” version will start at the first address location and automatically terminate at the given end address.

At any point “x” can be entered as a mnemonic and **as** will terminate and return the user to the Mbug prompt.

Details of the Assembler syntax are documented in CHAPTER 4 : SINGLE LINE ASSEMBLER/DISASSEMBLER.

Examples:

```
Mbug>>as 0+
0x00000000 0x0000 bkpt          >> movi R2,31
0x00000002 0x0000 bkpt          >> addi r1,r2
0x00000004 0x0000 bkpt          >> ld r2,(r1,0)
0x00000006 0x0000 bkpt          >> bkpt
0x00000008 0x0000 bkpt          >> x
Mbug >>
```

bd

Breakpoint Delete

brdel address

brdel will delete a breakpoint that was previously set at a specified address in memory. This command will remove one breakpoint at a time. If several breakpoints must be deleted, this command must be executed once for each address.

This command has been made obselete by the "nobr" command introduced in Version 1.5 and may be removed in later releases.

Examples:

```
Mbug >>brset 3400
breakpoint set at 0x00003400
Mbug >>brset 3420
breakpoint set at 0x00003420
Mbug >>brlist
Current Breakpoint List:
        0x00003400
        0x00003420
Mbug >>brdel 3420
breakpoint deleted from 0x00003420
Mbug >>brlist
Current Breakpoint List:
        0x00003400
Mbug >>
```

bl

List Breakpoints

bl

The breakpoints that are currently set will be displayed on the terminal.

This command has been made obselete by the "br" command introduced in Version 1.5 and may be removed in later releases.

Examples:

```
Mbug >>brlist
Current Breakpoint List:
Mbug >>brset 3400
breakpoint set at 0x00003400
Mbug >>brlist
Current Breakpoint List:
        0x00003400
Mbug >>brset 3420
breakpoint set at 0x00003420
Mbug >>brlist
Current Breakpoint List:
        0x00003400
        0x00003420
Mbug >>
```

br

Set a Breakpoint

br **address**

br

br will set a breakpoint at a specified address in memory. Breakpoint set will not remove a breakpoint from an address if a breakpoint already exists there.

This command will set one breakpoint at a time. If several breakpoints must be set, this command must be executed once for each address. A maximum of 20 breakpoints can be set in the system.

br will list all currently set breakpoints when invoked without the address argument.

Examples:

```
Mbug >>br
Current Breakpoint List:
Mbug >>br 3400
breakpoint set at 0x00003400
Mbug >>br
Current Breakpoint List:
        0x00003400
Mbug >>br 3420
breakpoint set at 0x00003420
Mbug >>br
Current Breakpoint List:
        0x00003400
        0x00003420
Mbug >>
```

bs

Set a Breakpoint

brset address

brset will set a breakpoint at a specified address in memory. Breakpoint set will not remove a breakpoint from an address if a breakpoint already exists there.

This command will set one breakpoint at a time. If several breakpoints must be set, this command must be executed once for each address. A maximum of 20 breakpoints can be set in the system.

This command has been made obselete by the "br" command introduced in Version 1.5 and may be removed in later releases.

Examples:

```
Mbug >>brlist
Current Breakpoint List:
Mbug >>brset 3400
breakpoint set at 0x00003400
Mbug >>brlist
Current Breakpoint List:
        0x00003400
Mbug >>brset 3420
breakpoint set at 0x00003420
Mbug >>brlist
Current Breakpoint List:
        0x00003400
        0x00003420
Mbug >>
```

da

Define Alias

da

This command will allow the user to define an alias to a list of commands (separated by a semicolon). Once the alias has been defined, Run Alias (**ra)** can be used instead of retyping the list of commands. Only one alias may be set at a time, and using Define Alias a second time will overwrite the previously aliased command list.

Below is an example of using an alias to define a combination of registers and memory to be displayed each time the alias is run.

Example:

```
Mbug >>da
Current alias definition:

New alias : rd r a epsr epc; md 1000
Alias defined as : rd r a epsr epc; md 1000

Mbug >>ra
r00:  0x00000000        r01:  0x0000ff00
r02:  0x00000000        r03:  0x00000000
r04:  0x00000000        r05:  0x00000000
r06:  0x00000000        r07:  0x00000000
r08:  0x00000000        r09:  0x00000000
r10:  0x00000000        r11:  0x00000000
r12:  0x00000000        r13:  0x00000000
r14:  0x00000000        r15:  0x00000000

a00:  0x00000000        a01:  0x0000ff00
a02:  0x00000000        a03:  0x00000000
a04:  0x00000000        a05:  0x00000000
a06:  0x00000000        a07:  0x00000000
a08:  0x00000000        a09:  0x00000000
a10:  0x00000000        a11:  0x00000000
a12:  0x00000000        a13:  0x00000000
a14:  0x00000000        a15:  0x00000000

epsr : 0x00000000
epc : 0x00000000
0x00001000      00000000 00000000 00000000 00000000     ................
Mbug >>
```

dl

Download Data From the Host

dl

This instruction provides the ability to receive data from the host serial port. The data can be received in two formats: S-Records or compressed S-Records. The firmware will automatically detect the format and decompress the data if necessary. The data which is downloaded will be placed in memory locations specified by the input file. See Chapter ?? for more information on compression and decompression.

The S-record checksums are ignored during downloads.

Example :

```
Mbug >> tm
```

(On the host machine (Unix):)

```
Unix $ ls
hello.c
hello.o
hello.srec
a.out
Unix $ cat hello.srec (<ctrl>-a)
```

(On the target board running Mbug:)

```
Mbug >> dl
Download Complete.
Mbug >>
```

Example :

```
Mbug >> dl
```

(On the host machine (PC):)

Use the terminal emulator's pull-down menu's to initiate transfer of the S-record file.Transfer mode should be text or ASCII for S-Records.

(On the target board running Mbug:)

```
Download Complete.
Mbug >>
```

ds

Disassembler

ds **address**
ds **start +**
ds **start - end**

The disassembler for the Mbug system displays the contents of memory at the given address. The contents are shown in hexadecimal opcode format as well as in M.CORE assembly instruction format.

If the “plus” form is used, the command goes into interactive mode and will continue reading and disassembling until terminated by the user. If the “range” form is used, the command will continue reading and disassembling for each inclusive address in the range specified.

The different forms can be combined in a single command by separating the forms with a comma or whitespace. This will display multiple disassembled portions of the memory space with one command.

Examples:

```
Mbug >>ds 0,4
0x00000000 0x61f2 movi    r2,0x1f
0x00000004 0x8201 ld      r1,(r2,0x0)
Mbug >>ds 0
0x00000000 0x61f2 movi    r2,0x1f
Mbug >>ds 0+
0x00000000 0x61f2 movi    r2,0x1f
0x00000002 0x2011 addi    r1,0x2
0x00000004 0x8201 ld      r1,(r2,0x0)
0x00000006 0x0000 bkpt
0x00000008 0x0000 bkpt
0x0000000a 0x0000 bkpt
0x0000000c 0x0000 bkpt
0x0000000e 0x0000 bkpt
0x00000010 0x0000 bkpt
0x00000012 0x0000 bkpt
0x00000014 0x0000 bkpt
0x00000016 0x0000 bkpt
0x00000018 0x0000 bkpt
0x0000001a 0x0000 bkpt
0x0000001c 0x0000 bkpt
0x0000001e 0x0000 bkpt
x to quit, anything else to continue >> x
Mbug >>ds 2-4
0x00000002 0x2011 addi    r1,0x2
0x00000004 0x8201 ld      r1,(r2,0x0)
Mbug >>
```

go

Execute User Code

go **address**

go +

This command allows the user to initiate execution of code starting at a given address. An rte instruction is used to transfer control from the monitor to user code. The **go** command copies the start address into the EPC (Exception Program Counter) register (if supplied). The rte instruction then loads the PC from the EPC and the PSR from the EPSR (Exception Processor Status Register). The user must have previously written EPSR to the desired value.

The “plus” form will continue execution at the address already contained in the EPC register. This is useful for continuing where a breakpoint had previously stopped execution.

Control returns to the monitor whenever the user program encounters an exception, provided the user program has not rewritten the vector table or Vector Base Register (VBR). The address of the instruction currently executing will be written to the screen as well as the disassembled instruction at that address.

Examples:

```
Mbug >> ds 4de-4ec
0x000004de 0x6004 movi    r4,0x0
0x000004e0 0x6325 movi    r5,0x32
0x000004e2 0x8604 ld      r4,(r6,0x0)
0x000004e4 0x1c56 addu    r6,r5
0x000004e6 0x9604 st      r4,(r6,0x0)
0x000004e8 0x2003 addi    r3,0x1
0x000004ea 0x0f23 cmpne   r3,r2
0x000004ec 0xeff8 bf      0x000004e2
Mbug >> br 4e6
breakpoint set at 0x000004e6
Mbug >> go 4de
A Program exception has occurred.
Breakpoint Encountered:
Current Instruction Pointer: 0x000004e6 st      r4,(r6,0x0)
Mbug >> go +
A Run Mode or Trace exception has occurred.
A Program exception has occurred.
Breakpoint Encountered:
Current Instruction Pointer: 0x000004e6 st      r4,(r6,0x0)
Mbug >>
```

help he

Help on Mbug Commands

help **command**

he **command**

This provides information on the commands implemented by Mbug. For a full list of commands see **menu**.

Examples:

```
Mbug >>help go
GO:
===
Syntax: go <address>
        go +

Description:

This command allows the user to execute user code
starting at <address>. If the "+" argument is used, then
the execution starting point is defined by the contents of
bits 31-1 of the user's EPC.

Note that if a breakpoint is encountered, then entering
"go +" will allow execution to continue where the
breakpoint stopped it.
Mbug >>help md
MEMORY DISPLAY:
==============
Syntax: md <address>[{-<address>,+}]
        mdh <address>[{-<address>,+}]
        mdb <address>[{-<address>,+}]

Description:

Displays data stored in the given memory location(s).
The display will always start aligned to the 16-byte boundary
containing the supplied starting address. Addresses must be aligned
to the size of the displayed data. The width of the memory accesses
corresponds to the width of the displayed data. 'md' displays four
32-bit words per line formatted as 8 hex digits each; 'mdh' displays 8
16-bit halfwords per line formatted as 4 hex digits each; and, 'mdb'
displays 16 8-bit bytes per lineformatted as 2 hex digits each.
The ASCII representation of the data is also printed in each case.

If the specified address range exceeds 64 bytes, the command will enter
an interactive mode, pausing after each four lines of displayed data. At
each pause the user can select whether to continue or exit the command early.
A <cr> signals the next four lines to be displayed. This sequence repeats
until the entire memory range has been displayed, or the user enters an 'x'
at the prompt
Mbug >>
```

log

Toggles Transcript Logging

log

This command provides the capability to log a debug session. The command toggles the logging function. When logging is enabled, all characters sent to the terminal will be echoed to the host port.

The log function is automatically disabled if Set Input (**si**) has been used to indicate that all I/O (terminal interaction as well as downloads) will go through the keyboard serial port.

Example:

```
Mbug >> log
You are enabling logging! After this message all input and output to your
terminal will be mirrored out to the host port. Now would be a time to open an
editor on the host and get into insert mode
Mbug >> log
Logging disabled!
Mbug >>
```

md   md

Display Memory Word

md **address**
md **start +**
md **start - end**

This command will display memory as 4 words per line of output. If the “range” form is used and it exceeds 16 words then memory display will enter interactive mode. In interactive mode, the user needs to type <return> to continue or “x” to exit. The start address is normalized to the previous quad-word boundary. Likewise, the ending address is normalized to the next quad-word boundary. For example, if the start address was 0x00000104 then the first memory address to be displayed would be 0x00000100. If the end address was 0x00000104 then the last location to be displayed would be 0x0000010C. All address arguments must be word aligned.

The “plus” form will cause the command to enter interactive mode and display memory in 16-word blocks from a given start address and continue until the user types “x”.

Note that the above parameter forms can be combined by separating the forms with a comma or whitespace. This will display multiple portions of the memory space with one command.

Examples:

```
Mbug >> md 50100,50200
0x00050100  00000041 00000042 00000043 00000044         ...A...B...C...D
0x00050200  00000000 00000000 00000000 00000000         ................
Mbug >> md 50100-50130
0x00050100  00000041 00000042 00000043 00000044         ...A...B...C...D
0x00050110  00000045 00000046 00000047 00000048         ...E...F...G...H
0x00050120  00000000 00000000 00000000 00000000         ................
0x00050130  00000000 00000000 00000000 00000000         ................
Mbug >> md 50250+
0x00050250  40000040 00000000 00000000 00000000         @..@............
0x00050260  00000000 00000000 00000000 00000000         ................
0x00050270  00000000 00000000 00000000 00000000         ................
0x00050280  00000000 00000000 00000000 24002400         ............$.$.
md >>
0x00050290  00000000 00000000 00000000 00000000         ................
0x000502a0  2a2a2a2a 00000000 00000000 00000000         ****............
0x000502b0  00000000 00000000 00000000 00000000         ................
0x000502c0  00000000 00000000 00000000 00000000         ................
md >>x
Mbug >>
```

mdh mdh

Display Memory Halfword

mdh address
mdh start +
mdh start - end

This command will display memory as 8 halfwords per line of output. If the “range” form is used and it exceeds 32 halfwords then memory display will enter interactive mode. In interactive mode, the user needs to type <return> to continue or “x” to exit. The start address is normalized to the previous quad-word boundary. Likewise, the ending address is normalized to the next quad-word boundary. For example, if the start address was 0x00000104 then the first memory address to be displayed would be 0x00000100. If the end address was 0x00000104 then the last location to be displayed would be 0x0000010C. All address arguments must be halfword aligned.

The “plus” form will cause the command to enter interactive mode and display memory in 16-word blocks from a given start address and continue until the user types “x”.

Note that the above parameter forms can be combined by separating the forms with a comma or whitespace. This will display multiple portions of the memory space with one command.

Examples:

```
Mbug >>mdh 100,200
0x00000100   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00000200   0000 0000 0000 0000 0000 0000 0000 0000    ................
Mbug >>mdh 1000-1030
0x00001000   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001010   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001020   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001030   0000 0000 0000 0000 0000 0000 0000 0000    ................
Mbug >>mdh 1000-1080
0x00001000   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001010   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001020   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001030   0000 0000 0000 0000 0000 0000 0000 0000    ................
mdh >>
0x00001040   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001050   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001060   0000 0000 0000 0000 0000 0000 0000 0000    ................
0x00001070   0000 0000 0000 0000 0000 0000 0000 0000    ................
mdh >>x
0x00001080   0000 0000 0000 0000 0000 0000 0000 0000    ................
Mbug >>
```

mdb mdb

Display Memory Byte

mdb address
mdb start +
mdb start - end

This command will display memory as 16 bytes per line of output. If the “range” form is used and it exceeds 16 words then memory display will enter interactive mode. In interactive mode, the user needs to type <return> to continue or “x” to exit. The start address is normalized to the previous quad-word boundary. Likewise, the ending address is normalized to the next quad-word boundary. For example, if the start address was 0x00000104 then the first memory address to be displayed would be 0x00000100. If the end address was 0x00000104 then the last location to be displayed would be 0x0000010C.

The “plus” form will cause the command to enter interactive mode and display memory in 16-word blocks from a given start address and continue until the user types “x”.

Note that the above parameter forms can be combined by separating the forms with a comma or whitespace. This will display multiple portions of the memory space with one command.

Examples:

```
Mbug >>mdb 1000,3000
0x00001000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
Mbug >>mdb 2000-2040
0x00002000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00002010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00002020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00002030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00002040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
Mbug >>mdb 3000+
0x00003000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
mdb >>
0x00003040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
mdb >>x
0x00003080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x00003090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x000030a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x000030b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

```
Mbug >>
```

menu   me

Show Mbug Command List

menu

me

?

This command will list all of the commands that are available in the current implementation of Mbug.

Examples:

```
Mbug >> menu
COMMAND LIST
Command                  Mnemonic      Command                  Mnemonic
=======                  ========      =======                  ========
About...                 about         Memory Fill              mf
Assemble                 as            Memory Modify            mm
Breakpoint Set           br            Memory Modify Halfword   mmh
Breakpoint List          br            Memory Modify Byte       mmb
Breakpoint Delete        nobr          Memory Move              mv
Define Alias             da            Memory Search            ms
Disassemble              ds            Register Display         rd
Download                 dl            Register Modify          rm
Go                       go            Run Alias                ra
Help                     help          Set Input                si
Log session              log           Transparent Mode         tm
Menu                     menu or '?'   Trace                    tr
Memory Display           md            Verify download          ver
Memory Display Halfword  mdh
Memory Display Byte      mdb           Repeat last command      '.'

For additional details, type 'help <command>'
Mbug >>
```

mf
Memory Fill

mf start end data

The range of memory spanning from the starting address to the ending address is filled in with the given 32-bit data pattern. The fill is inclusive of the end point. Address arguments must be word aligned.

Examples:

```
Mbug >> mf 50100 50200 89898989
Mbug >> mf 50140 5015c 00000000
Mbug >> md 50120-50160
0x00050120  89898989 89898989 89898989 89898989         ................
0x00050130  89898989 89898989 89898989 89898989         ................
0x00050140  00000000 00000000 00000000 00000000         ................
0x00050150  00000000 00000000 00000000 00000000         ................
0x00050160  89898989 89898989 89898989 89898989         ................
Mbug >> mf 50144 50144 44444444
Mbug >> md 50120-50160
0x00050120  89898989 89898989 89898989 89898989         ................
0x00050130  89898989 89898989 89898989 89898989         ................
0x00050140  00000000 44444444 00000000 00000000         ....DDDD........
0x00050150  00000000 00000000 00000000 00000000         ................
0x00050160  89898989 89898989 89898989 89898989         ................
Mbug >>
```

mm

Memory Modify Word

mm address
mm start +
mm start - end

Memory modify is an interactive command. It will display the contents of the given memory address and allow the user to change the value stored there. Memory is considered to be a contiguous set of 32-bit integers. Addresses must be word aligned.

The “plus” form causes the command to start at a given address and continue until the end of memory or until the user types “x” to exit the memory modify loop.

The “range” form allows modifications for the inclusive range from start to end. When the end address is reached the memory modify loop is automatically exited. The user can type “x” at any time to exit the memory modify loop.

Examples:

```
Mbug >> memod 50100

     0x00050100 : 0x89898989 : ?  44444444

Mbug >> memod 50110-50118

     0x00050110 : 0x89898989 : ?  11111111
     0x00050114 : 0x89898989 : ?  22222222
     0x00050118 : 0x89898989 : ?  33333333

Mbug >> memod 50200+

     0x00050200 : 0x89898989 : ?  12341234
     0x00050204 : 0x00000000 : ?  12341234
     0x00050208 : 0x00000000 : ?  x

Mbug >> memod 50240-50270

     0x00050240 : 0x00000000 : ?  2133a234
     0x00050244 : 0x00000000 : ?  ffdfaddf
     0x00050248 : 0x00000000 : ?  x

Mbug >>
```

mmh mmh

Memory Modify Halfword

mmh address
mmh start +
mmh start - end

Memory modify halfword is an interactive command. It will display the contents of the given memory address and allow the user to change the value stored there. Memory is considered to be a contiguous set of 16-bit integers. Addresses must be half-word aligned.

The “plus” form causes the command to start at a given address and continue until the end of memory or until the user types “x” to exit the memory modify loop.

The “range” form allows modifications for the inclusive range from start to end. When the end address is reached the memory modify loop is automatically exited. The user can type “x” at any time to exit the memory modify loop.

Examples:

```
Mbug >>mmh 3400

     0x00003400 : 0x0000 : >>  1234

Mbug >>mmh 3400-3406

     0x00003400 : 0x1234 : >>  8989
     0x00003402 : 0x0000 : >>  aaaa
     0x00003404 : 0x0000 : >>  5555
     0x00003406 : 0x0000 : >>  4321

Mbug >>mmh 3400+

     0x00003400 : 0x8989 : >>  3333
     0x00003402 : 0xaaaa : >>  1111
     0x00003404 : 0x5555 : >>  6666
     0x00003406 : 0x4321 : >>  8888
     0x00003408 : 0x0000 : >>  1234
     0x0000340a : 0x0000 : >>  x

Mbug >>
```

mmb mmb

Memory Modify Byte

mmb address
mmb start +
mmb start - end

Memory modify byte is an interactive command. It will display the contents of the given memory address and allow the user to change the value stored there. Memory is considered to be a contiguous set of 8-bit integers. There are no alignment restrictions on the addresses.

The “plus” form causes the command to start at a given address and continue until the end of memory or until the user types “x” to exit the memory modify loop.

The “range” form allows modifications for the inclusive range from start to end. When the end address is reached the memory modify loop is automatically exited. The user can type “x” at any time to exit the memory modify loop.

Examples:

```
Mbug >>mmb 3400

     0x00003400 : 0x33 : >>  12

Mbug >>mmb 3400-3405

     0x00003400 : 0x12 : >>  55
     0x00003401 : 0x33 : >>  aa
     0x00003402 : 0x11 : >>  89
     0x00003403 : 0x11 : >>  fa
     0x00003404 : 0x66 : >>  23
     0x00003405 : 0x66 : >>  65

Mbug >>mmb 3400+

     0x00003400 : 0x55 : >>  ee
     0x00003401 : 0xaa : >>  45
     0x00003402 : 0x89 : >>  32
     0x00003403 : 0xfa : >>  x

Mbug >>
```

ms

ms

Memory Search

ms **start end data**

This command will search the block of memory from start to end for the specified (word-size) data. All matches within the range will be reported. If no match is found, nothing is printed. Leading zeros are automatically added to determine the full search word. Both start and end addresses must be word aligned.

Examples:

```
Mbug >> md 50120-50160
0x00050120  44000044 00000000 00000000 00000000         D..D............
0x00050130  00000000 00000000 00000000 00000000         ................
0x00050140  00004400 00000000 12340000 00000000         ..D......4......
0x00050150  00000000 00001234 12340000 00000044         .......4.4.....D
0x00050160  00000000 00000000 00000000 00000000         ................
Mbug >> ms 50120 50160 12340000
        0x00050148
        0x00050158
Mbug >> ms 50120 50160 44
        0x0005015c
Mbug >> ms 50120 50160 1234
        0x00050154
Mbug >>
```

mv mv

memory move

mv **start end dest**

This command will move the block of memory between start and end to a destination block of memory of the same size starting at dest. All three addresses must be word aligned. Data is transferred as 32-bit quantities.

Examples:

```
Mbug >> mf 50100 50110 ffffffff
Mbug >> md 50100-50150
0x00050100      ffffffff ffffffff ffffffff ffffffff         ................
0x00050110      ffffffff 00000000 00000000 00000000         ................
0x00050120      00000000 00000000 00000000 00000000         ................
0x00050130      00000000 00000000 00000000 00000000         ................
0x00050140      00000000 00000000 00000000 00000000         ................
0x00050150      00000000 00000000 00000000 00000000         ................
Mbug >> mv 50100 50110 50140
Mbug >> md 50100-50150
0x00050100      ffffffff ffffffff ffffffff ffffffff         ................
0x00050110      ffffffff 00000000 00000000 00000000         ................
0x00050120      00000000 00000000 00000000 00000000         ................
0x00050130      00000000 00000000 00000000 00000000         ................
0x00050140      ffffffff ffffffff ffffffff ffffffff         ................
0x00050150      ffffffff 00000000 00000000 00000000         ................
Mbug >>
```

nobr

No Breakpoint

nobr **address**

nobr

The No breakpoint command (**nobr**) will delete a breakpoint that was previously set at a specified address in memory. This command will remove one breakpoint at a time. If several breakpoints (but not all) are to be deleted, this command must be executed once for each address. Address arguments must be halfword aligned.

All currently set breakpoints are removed by invoking **nobr** without the address argument.

Examples:

```
Mbug >>br 3400
breakpoint set at 0x00003400
Mbug >>br 3420
breakpoint set at 0x00003420
Mbug >>br 3440
breakpoint set at 0x00003440
Mbug >>br
Current Breakpoint List:
      0x00003400
      0x00003420
      0x00003440
Mbug >>nobr 3420
breakpoint deleted from 0x00003420
Mbug >>br
Current Breakpoint List:
      0x00003400
      0x00003440
Mbug >>nobr
All breakpoints removed.
Mbug >>br
Current Breakpoint List:
Mbug >>
```

rd

Register Display

rd	**r**	- entire general register family
rd	**rx**	- one general purpose register
rd	**rx+**	- from rx to r15
rd	**rx-ry**	- from rx to ry
rd	**a**	- entire alternate general register family
rd	**ax**	- one alternate general register
rd	**ax+**	- from ax to a15
rd	**ax-ay**	- from ax to ay
rd	**crx**	- one control register

This command will display the contents of the specified registers, while providing the user with several options for specifying the registers to be displayed. The whole family of general purpose registers or alternate general registers can be viewed by typing "**rd r**" or "**rd a**" respectively. A single register can be viewed by specifying rx, ax, or crx, where the first character(s) denotes the register family and the *x* denotes the register number. Control registers may be selected by their standard abbreviations as well as their register number.

The "plus" form displays the contents of the register family starting with the given register up to and including the last register in that family.

The "range" form displays the contents of the registers from *x* to *y*.

Note that the "entire family", "plus", and "range" forms are not available in the control register family.

The above parameter forms can be combined by separating them with a comma or whitespace. This will display multiple registers in different register families with one command. Note that the general and alternate register displays are aligned on an even-numbered register boundary, so if an even numbered register needs to be displayed, the odd-numbered register following it is also displayed.

Examples:

```
Mbug >>rd r4+
r04:  0x00000000        r05:  0x00000000
r06:  0x00000000        r07:  0x00000000
r08:  0x00000000        r09:  0x00000000
r10:  0x00000000        r11:  0x00000000
r12:  0x00000000        r13:  0x00000000
r14:  0x00000000        r15:  0x00000000
```

```
Mbug >>rd psr
psr : 0x00000000
Mbug >>rd r a4-a7 cr3 ss0
r00:  0x00000000        r01:  0x0000ff00
r02:  0x00000000        r03:  0x00000000
r04:  0x00000000        r05:  0x00000000
r06:  0x00000000        r07:  0x00000000
r08:  0x00000000        r09:  0x00000000
r10:  0x00000000        r11:  0x00000000
r12:  0x00000000        r13:  0x00000000
r14:  0x00000000        r15:  0x00000000

a04:  0x00000000        a05:  0x00000000
a06:  0x00000000        a07:  0x00000000

fpsr : 0x00000000
ss0 : 0x00000000
Mbug >>
```

rm

Register Modify

rm	**r**	- entire general register family
rm	**rx**	- one general purpose register
rmt	**rx+**	- from rx to r15
rm	**rx-ry**	- from rx to ry
rm	**a**	- entire alternate general register family
rm	**ax**	- one alternate general register
rm	**ax+**	- from ax to a15
rm	**ax-ay**	- from ax to ay
rm	**crx**	- one control register

This will display the contents of the specified registers and allow the user to modify them. There are several input formats for viewing and modifying the registers. To view and modify the entire family of general purpose registers or alternate general registers, the user can type “**rm r**” or “**rm a**” respectively. When this format is used, all of the registers of the respective family will be displayed on the screen one at a time beginning with register 0 and ending when register 15 is reached or when the user types “x”.

A single register can be viewed and modified by specifying rx, or ax, or crx, where the first character(s) denotes the register family and the *x* denotes the register number. Control registers may be selected by their standard abbreviations as well as their register number.

The “plus” form is used to display the contents of the register starting with the given register and ending when register 15 is reached or when the user types “x”.

The “range” form displays the contents of the registers from *x* to *y*.

Note that the “entire family”, “plus”, and “range” forms are not available in the control register family.

Note that the above parameter forms can be combined by separating them with a comma or whitespace. This will modify registers in different register families with one command.

Starting in Version 1.5, these command mnemnic has been shortened from **rmm** and **regmodmult** to **rm**.

Example:

```
Mbug >>rm r
     r00 = 0x00000000 : rm >>12345678
     r01 = 0x0000ff00 : rm >>87654321
     r02 = 0x00000000 : rm >>1
     r03 = 0x00000000 : rm >>x

Mbug >>rm r2-r4
     r02 = 0x00000001 : rm >>aaaa5555
     r03 = 0x00000000 : rm >>5555aaaa
     r04 = 0x00000000 : rm >>99999999

Mbug >>rm r14+
     r14 = 0x00000000 : rm >>ffff00ff
     r15 = 0x00000000 : rm >>11001111

Mbug >>rm cr2, a5
epsr : 0x00000000

 new value ? rmm >>80008000

     a05 = 0x00000000 : rm >>fefefefe

Mbug >>rm a15
     a15 = 0x00000000 : rm >>12345678

Mbug >>rm r a cr2
r00:  0x12345678        r01:  0x87654321
r02:  0xaaaa5555        r03:  0x5555aaaa
r04:  0x99999999        r05:  0x00000000
r06:  0x00000000        r07:  0x00000000
r08:  0x00000000        r09:  0x00000000
r10:  0x00000000        r11:  0x00000000
r12:  0x00000000        r13:  0x00000000
r14:  0xffff00ff        r15:  0x11001111

a00:  0x00000000        a01:  0x0000ff00
a02:  0x00000000        a03:  0x00000000
a04:  0x00000000        a05:  0xfefefefe
a06:  0x00000000        a07:  0x00000000
a08:  0x00000000        a09:  0x00000000
a10:  0x00000000        a11:  0x00000000
a12:  0x00000000        a13:  0x00000000
a14:  0x00000000        a15:  0x12345678

epsr : 0x80008000
Mbug >>
```

ra

Run Alias

ra

This instruction will read in the string which the user has defined as an alias. Then, the commands in this string will be executed sequentially.

Example:

```
Mbug >> ra
```

The runalias command can also be embedded within a command line. For example, if the alias string has previously been defined as

```
tr +; rd r
```

Typing the command

```
Mbug >> log; trace 2100; ra; log
```

is identical to typing

```
Mbug >> log; trace 2100; tr +; rd r; log
```

See Define Alias (**da**) for a complete example on defining the alias.

si

si

Set serial I/O Configuration

si **both**
si **key**
si **?**

This is used to tell the firmware (Mbug) which port to use as the serial i/o.

key All i/o (terminal interaction as well as downloads) will go through the keyboard serial port. This allows the user to connect an evaluation board equipped with Mbug to a single serial line (for example a PC or Unix Workstation) without losing any functionality.

both Terminal interaction will go through the keyboard serial port while host interaction (downloads) will go through the host serial port.

? This will return one of the arguments described above, indicating the current setting.

The default is **key**.

The log and transparent mode functions are automatically disabled if Set Input has been used to indicate that all I/O will go through the keyboard serial port.

Example:

```
Mbug >> si ?
Set Input Port : Both Ports Active
Mbug >> si key
Set Input Port : set to Keyboard Port
Mbug >> si both
Set Input Port : set to Both Ports
Mbug >>
```

tr tr

Single Step Trace

tr **address**

tr +

This command allows the user to single-step through a user program. The microprocessor will execute a single instruction, and then return control back to the firmware. If a specific address is given, then a single instruction is executed from that address. However, if the "plus" form is used, then the address of the instruction to execute is derived from the EPC (Exception Program Counter) register. After the instruction has been executed, control is returned to the firmware (Mbug) and the user can examine the programming model or continue to trace through instructions.

An rte instruction is used to transfer control from the monitor to user code. The Trace command copies the start address into the EPC (Exception Program Counter) register (if supplied). The rte instruction then loads the PC from the EPC and the PSR from the EPSR (Exception Processor Status Register). The user must have previously written EPSR to the desired value.

The "plus" form will continue execution at the address already contained in the EPC register. This is useful for tracing through a sequence of instructions.

Example:

```
Mbug >> disassem 2100
0x00000000 0x1011 mfcr    r1,cr1
Mbug >> trace 2100
A Run Mode or Trace exception has occurred.
Current Instruction Pointer: 0x00002102 0x0064 ldm     r4-r15,(r0)
Mbug >> trace +
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002104 0xbeef st.b    r15,(r14,0xe)
Mbug >> .
A Run Mode or Trace exception has occurred.
Current instruction Pointer: 0x00002106 0x1021 mfcr    r1,cr2
Mbug >>
```

tm

Enter Transparent Mode

tm

This command will put Mbug into a transparent mode. In other words, as the user types data into the keyboard, the data is sent directly to the host serial port. In addition, data that comes in from the host serial port will be forwarded to the keyboard serial port. This allows the target board to speak directly to the host system. Very often, the host machine is a Unix system and using this command will provide access to a Unix operating system for use along with the target board. This command is also used for retrieving downloadable files. The user can break out of transparent mode by typing <ctrl>-a. (To use this feature, the Set Input setting must be both).

Example:

```
Mbug >> tm
```

(On the host machine (Unix):)

```
Unix $ ls
hello.c
hello.o
hello.srec
Unix $ sleep 1; cat hello.srec(<ctrl>-a)
```

(On the target board running Mbug:)

```
Mbug >> dl
Download Complete.
Mbug >>
```

ver

Verify Data From The Host

ver

This command provides the ability to verify data received from the host serial port. The data can be received in S-record format. The data is compared on a byte by byte basis against memory locations specified by the input file. The first mismatch found will terminate the command with an error report providing the address, expected data and actual data.

Verify calculates and validates the checksum information provided in the downloaded file. Any mismatch terminates the command and the error is reported to the terminal.

Example :

```
Mbug >> tm
```

(On the host machine (Unix):)

```
Unix $ ls
hello.c
hello.o
hello.srec
a.out
Unix $ cat hello.srec (<ctrl>-a)
```

(On the target board running Mbug:)

```
Mbug >> ver
Verify Complete.
Mbug >>
```

Example :

```
Mbug >> ver
```

(On the host machine (PC):)

Use the terminal emulator’s pull-down menu’s to initiate transfer of the S-record file.Transfer mode should be text or ASCII for S-Records.

(On the target board running Mbug:)

```
Verify Complete.
Mbug >>
```

CHAPTER 4 : SINGLE LINE ASSEMBLER/ DISASSEMBLER

4.1 Assembler Syntax

The single line assembler and disassembler follow the M.CORE Application Binary Interface (ABI) standard wherever possible. There are some exceptions, however due to the nature of single line assembly versus file based assembly. The following subsections detail the

4.1.1 Labels

Labels are not allowed by the single line assembler and will be flagged as a syntax error.

4.1.2 Register Specifiers

General purpose registers are specified as rn where n is an integer in the range of 0 through 15 inclusive. No distinction is made between the general and alternate registers files.

Control Registers are specified as crn where n is a number from 0 through 31 inclusive. Unimplemented control registers are not flagged by the assembler. The single line assembler does not currently support the common names assigned to the control registers (eg., PSR is not recognized as an alias to cr0).

4.1.3 Immediate Constants

Immediate constants are assumed to be decimal (Base 10) unless preceded by "0x" indicating that they are hexadecimal (Base 16). Octal and character constants are not supported. Offset and modulo32 immediates (zero indicates a value of 32) follow the syntax specified in the M.CORE Programmer's Reference Manual. The assembler will adjust the immediate as appropriate before inserting it into the opcode.

Examples:

```
Mbug >>as 4000+
0x00004000 0x0000 bkpt          >> movi r4,45
0x00004002 0x0000 bkpt          >> movi r4,0x45
0x00004004 0x0000 bkpt          >> addi r3,32
0x00004006 0x0000 bkpt          >> bmaski r5,32
0x00004008 0x0000 bkpt          >> bmaski r5,0
ERROR : operand out of range
0x00004008 0x0000 bkpt          >> x
Mbug >>ds 4000-4008
0x00004000 0x62d4 movi    r4,0x2d
0x00004002 0x6454 movi    r4,0x45
0x00004004 0x21f3 addi    r3,0x20
0x00004006 0x2c05 bmaski  r5,0x20
0x00004008 0x0000 bkpt
Mbug >>
```

4.1.4 Load/Store Addressing

Load and Store instruction indirect address specifiers follow the syntax documented in the M.CORE Programmer's Reference Manual. The displacement values are specified in bytes;

however, for the halfword and word sized operations a misaligned offset will be truncated to the next lower aligned boundary. This operation is consistent with the scaled offset value stored in the opcode. The parenthesis surrounding the address specifier are required.

The default load/store size is word. Size modifiers are used to specify anything other than the default. “.b” (or, simply “b”) specifies a byte sized operation; “.h” or “h” a halfword sized operation; and, “.w” or “w” a word sized operation.

Example:

```
Mbug >>as 5000+
0x00005000 0x0000 bkpt >> ld.b r4,(r2,1)
0x00005002 0x0000 bkpt >> ldb r4,(r2,2)
0x00005004 0x0000 bkpt >> ld.h r4,(r2,1)
0x00005006 0x0000 bkpt >> ldh r4,(r2,2)
0x00005008 0x0000 bkpt >> ld.w r4,(r2,1)
0x0000500a 0x0000 bkpt >> ldw r4,(r2,2)
0x0000500c 0x0000 bkpt >> ld r4,(r2,4)
0x0000500e 0x0000 bkpt >> x
Mbug >>ds 5000-500e
0x00005000 0xa412 ld.b r4,(r2,0x1)
0x00005002 0xa422 ld.b r4,(r2,0x2)
0x00005004 0xc402 ld.h r4,(r2,0x0)
0x00005006 0xc412 ld.h r4,(r2,0x2)
0x00005008 0x8402 ld r4,(r2,0x0)
0x0000500a 0x8402 ld r4,(r2,0x0)
0x0000500c 0x8412 ld r4,(r2,0x4)
0x0000500e 0x0000 bkpt
Mbug >>
```

4.1.5 Branch Offsets

Branch offsets are one of the places where the single line assembler differs from the ABI. Branch offsets are to be specified as the actual target address. The assembler will calculate the offset based on the current memory address pointer and use the calculated value to construct the opcode.

Branch offsets default to decimal, unless overridden by the “0x”modifier to specify a hexadecimal value.

Example:

```
Mbug >>as 5000
0x00005000 0x0000 bkpt >> br 0x5000
Mbug >>ds 5000
0x00005000 0xf7ff br 0x00005000
Mbug >>
```

4.1.6 LRW Offsets

LRW offsets follow the same convention as branch offsets; the offset is entered as the actual target address and the assembler calculates the correct offset to put in the opcode. The value must be surrounded by square brackets (“[<value>]”) as documented in the M.CORE Programmer’s Reference Manual.

Example:

```
Mbug >>as 4000+
0x00004000 0x0000 bkpt >> lrw r6,[0x4000]
0x00004002 0x0000 bkpt >> br 0x4008
0x00004004 0x0000 bkpt >> x
Mbug >>mm 4004
0x00004004 : 0x00000000 : >> 12345678
Mbug >>ds 4000-4008
0x00004000 0x7600 lrw r6,[0x00004000]
0x00004002 0xf002 br 0x00004008
0x00004004 0x1234 mov r4,r3
0x00004006 0x5678 DATA 0x5678
0x00004008 0x0000 bkpt
Mbug >>
```

CHAPTER 5 : Mbug EXCEPTION SUPPORT

There are 17 exception traps in this version of Mbug. A message indicating which exception has occurred is displayed for all of them except System Reset. Table 3: *Exceptions Trapped and Identifed by Mbug* lists the exceptions trapped by Mbug Version 1.0. All vector addresses not specifically listed vector through a common reserved vector routine.

Table 3: Exceptions Trapped and Identifed by Mbug

Vector Offset	Exception Type
0x0000	System Reset
0x0004	Misaligned Access
0x0008	Access Error
0x000C	Divide by Zero
0x0010	Illegal Instruction
0x0014	Pivilege Violation
0x0018	Trace Exception
0x001C	Breakpoint Exception
0x0020	Unrecoverable Error
0x0024	Soft Reset
0x0028	Normal Interrupt Autovector
0x002C	Fast Interrupt Autovector
0x0030	Hardware Accelerator
0x0040	Trap 0 Instruction
0x0044	Trap 1 Instruction
0x0048	Trap 2 Instruction
0x004C	Trap 3 Instruction

System Reset occurs when the software is booted up or the evaluation board is reset. The other exceptions occur due to interrupts or errors in the execution of the code. For details on what causes each exception, see the M.CORE Programmer's Reference Manual.

For interrupt-driven I/O, an exception will also occur when <ctrl>-c is invoked by the

user. When <ctrl>-c is pressed, Mbug aborts the user code and gives control back to the firmware. The MMC2001 version of Mbug utilizes polled I/O, therefore the <ctrl>-c interrupt feature is not available.

The user is notified of an exception by a message that appears on the terminal. Control is returned to the firmware. If the exception was caused by the completion of a trace or by arriving at a breakpoint during execution of the user's code, the user can continue testing. Otherwise the user may need to modify the code to correct a problem and download the program again to resume testing.

CHAPTER 6 : RESTRICTIONS

6.1 Mbug use of Control Registers

There are five Supervisor Scratch (ss0-ss4) registers, numbered 0 through 4. Mbug makes use of ss3 and ss4, so any user values placed into these two registers will be destroyed whenever control is returned to the monitor. The user is encouraged to place any values that are of interest or necessity into only ss0 through ss2, although the user can use the other two scratch registers for calculations or temporary storage.

6.2 Interrupt Driven I/O

Interrupt driven I/O is not active in Version 1.0 of Mbug. Some code has been written for this functionality, but it has not been tested. Therefore, interruption of execution using <ctrl>-c is not supported.

6.3 Memory Test

No diagnostic memory tests have been included in Mbug. The user must incorporate tests that will provide this function, if desired.

Appendix A : Mbug Command Summary

Command Mnemonic	Command Summary
.	Repeat last command
as	Assemble instruction at address(es)
nobr	Delete breakpoint at address
br	List breakpoints
br	Set breakpoint at address
da	Define alias for command sequence
ds	Disassemble an instruction at address(es)
dl	Download S-Record file to memory
go	Execute from address
he	Show more information on command
log	Record debug session to host
md	Display memory as words
mdh	Display memory as halfwords
mdb	Display Memory as bytes
mf	Fill memory block with data pattern
mm	Modify word at memory address(es)
mmh	Modify halfword at memory address(es)
mmb	Modify byte at memory address(es)
mv	Move memory block to destination
ms	Search memory block for data word
me	Show list of available commands
rd	Display selected register(s)
rm	Modify selected register(s)
ra	Execute commands in the alias
si	Set I/O port
tr	Trace instructions from given address
tm	Enter transparent mode to host
ver	Verify memory with S-records

Appendix B : Mbug Auxillary Commands

Mbug provides an auxillary set of non-interactive commands for use by debuggers and other programs which must communicate with the debugger in a predictable fashion. These commands are summarized in the following table.

Command Mnemonic	Command Summary
as1	Assemble a Single Instruction
mr	Memory Read (Word)
mrh	Memory Read (Halfword)
mrb	Memory Read (Byte)
mw	Memory Write (Word)
mwh	Memory Write (Halfword)
mwb	Memory Write (Byte)
rw	Register Write

Appendix C : Mbug ERROR CODES AND MESSAGES

Parser Errors

```
0xfb00   UNKNOWN_COMMAND          unrecognized command
0xfb01   UNKNOWN_REGISTER         unknown register
0xfb02   UNKNOWN_CTL_REGISTER     unknown control register
0xfb03   ILLEGAL_RD_STAGE         cannot specify whole register family in
range
0xfb04   ILLEGAL_REG_FAMILY       cannot specify a range of control registers
0xfb05   RANGE_CROSS_FAMILY       cannot specify a range across register
families
0xfb06   UNIMPLEMENTED_STAGE      invalid rd or rmm parameter format
0xfb07   UNKNOWN_OPERATOR         unknown operator in arguments
0xfb08   INVALID_FILENAME         invalid download filename
```

Errors from Error Checking Toolbox

```
0xfd00   INVALID                  NOT valid
0xfd01   VALID                    valid
0xfd02   INVALID_SIZE             invalid address size
0xfd03   OUT_OF_BOUNDS_ADDRESS    address not in bounds
0xfd04   INVALID_HEX_INPUT        invalid hex address
0xfd05   INVALID_REGISTER         invalid register number
0xfd07   NOT_WORD_ALIGNED         not a word aligned address
0xfd06   NOT_HWORD_ALIGNED        not a half-word aligned address
0xfd08   REVERSED_ADDRESS         start address is greater than end
0xfd09   RANGE_OVERLAP            destination overlaps source addresses
0xfd0a   ERROR                    an unidentified error has occured
0xfd0b   INVALID_PARAM            invalid input parameter
```

Get Argument Errors

```
0xFE00   INVALID_NUMBER_ARGS      invalid number of arguments
0xFE01   UNKNOWN_PARAMETER        unknown type of command parameter
```

Tokenizer Toolbox Errors

```
0xFF00   ILLEGAL_CHARACTER        unrecognized character in input stream
0xFF01   TTL_NOT_SORTED           token translation list not sorted
0xFF02   TTL_NOT_DEFINED          token translation list not assigned
0xFF03   INVALID_STRING           unable to extract string from input stream
0xFF04   BUFFER_EMPTY             input buffer is empty
0xFF05   INVALID_MODE             input buffer is in an unrecognized mode
0xFF06   TOK_INTERNAL_ERROR       internal tokenizer error
0xFF07   TOO_MANY_IBS             too many open input buffers
0xFF08   NO_OPEN_IBS              no open input buffers

0xf400   INSTRUCTION_SYNTAX       instruction syntax incorrect
0xf401   OUT_OF_RANGE             operand out of range
0xf402   ILLEGAL_SEPARATOR        illegal separator or number of arguments
0xf403   ILLEGAL_REG_OPERAND      illegal register operand
```

Screen Toolbox Errors

```
0xfc00   RESERVED_WORD            used a reserved word as an argument
0xf404   NUMBER_CONVERSION        Number converison error
```

Breakpoint Errors

```
0xFA00   FULL_BPDS                 breakpoint data structure is full
```

Download Errors

```
0xf900   NOT_IN_S_RECORD_FORMAT   not in S-Record Format
0xf901   UNREC_RECORD_TYPE        unrecognized record type
0xf902   CONVERSION_ERROR         ascii to int conversion error
0xf903   INVALID_MEMORY           bad s-record memory address
```

Compression and Decompression Errors

```
0xf800   COMP_UNK_CHARACTER       unknown compressed character
0xf801   COMP_UNKNOWN_STATE       unknown binary state
0xf802   NOT_IN_COMPRESSED_FORMAT not in compressed S-Record format
0xf904   COMPARE_FAILED           S-record verify with memory failed
```

Serial I/O Handling Errors

```
0xf700   UNKNOWN_PORT_STATE       unrecognized serial port configuration
0xf500   TM_NEEDS_BOTH_PORTS      transparent mode needs two serial ports
```

Register Errors

```
0xf600   SPR_NOT_FOUND             cannot find in special register file
```

Appendix D : CMB1200 Evaluation System

D.1 Memory Map

The memory model for Mbug as configured for the CMB1200 Evaluation System is shown in Figure 2-1. Mbug Memory Model (MMC2001Internal ROM). The exception vectors are located within the address range of 0x0000 - 0x01ff. The Mbug code begins at address 0x200 and extends approximately to 0xffff. Initialized data variables are also stored in this ROM address range and copied to RAM during initialization. On-chip RAM runs from 0x3000000 through 0x30007fff. RAM addresses 0x3000000 through 0x300004fff (20k bytes) are reserved for the user, while 0x30005000 - 0x30007fff are used by the monitor. The monitor stack grows down from 0x30007ffc.

D.2 Reset Initialization

Mbug performs a complete initialization of the system in response to a reset from any of the reset sources. The following sections detail the configuration of the system after Mbug initialization.

D.2.1 Reset Source Indication

The source(s) of the hardware reset will be displayed as part of the banner written at the completion of the initialization sequence. The reset source is determined by reading the contents of the Reset Status Register.

Warning: The reset status bits are cleared by a write to the status register. The intialization sequence must write this register to setup the clock output; therefore, user code intiated by Mbug cannot read the reset source register and expect valid data.

D.2.2 Clocks

The clock output is enabled by Mbug with the source derived from the high-ref. clock. Other options can be selected by writing the control re gister through the memory modify command or user code.

D.2.3 Chip Selects

The chip select functions are initilized according to the standard memory configuration on the CMB1200 Evaluation System. Flash memory access time was assumed to be 90nS. RAM wait states were calculated for 70 nS access time memories. All memories are assumed to be 16-bits wide. CS3 was configured for a “generic” 8-bit peripheral with timing setup as loose as possible through the chip select options. All wait state values were calculated for an perating frequency of 32 MHZ. Table 5: *CMB1200 Evaluation System Chip Select Initialization* summarizes the options programmed for each of the chip selects.

Table 5: CMB1200 Evaluation System Chip Select Initialization

CHIP SELECT	ADDRESS RANGE	PORT SIZE	WAIT STATES	SUP./ USER	R/W	LATE OE	LATE CS	EDC
CS0 (FLASH)	`2d000000-2dffffff`	16	4	S/U	Read Only	No	No	No
CS1 (RAM)	`2f000000-2fffffff`	16	3	S/U	R/W	No	No	No
CS2 (RAM)	`2e000000-2effffff`	16	3	S/U	R/W	No	No	N
CS3 (PERIPH)	`2c000000-2fffffff`	8 (D0-D7)	15	S/U	R/W	Yes	Yes	Yes

D.2.4 Show Cycles

Mbug enables show cycles with the options to show all cycles at the cost of performance. The low order data bits are brought out. If the user needs to select other options, the control register can be written with a memory modify command, or from user code.

D.2.5 Internal RAM and ROM

Internal RAM and ROM are intialized so that both supervisor and user accesses are allowed.

D.2.6 User Registers

A virtual copy of the User's registers are stored in system memory. These registers are intialized during the reset intitialization sequence as described below. Note that these values are not actually copied into the CPU registers until user code is executed from the Go or Trace commands. A more detailed description of debugger operation and its relationship to the user can be found in Section ???.

The Processor Status Register (PSR) is initialized to supervisor mode, interrupts disabled, and exceptions enabled.

The Vector Base Register (VBR) is initialized to point at the beginning of the boot memory where Mbug's vectors are located. If the user wishes to move the vector table elsewhere, the trace exception (offset 0x18) and breakpoint exception (offset 0x1c) vectors should be copied from the Mbug table into the user's table. Without these two vectors, Mbug's trace and breakpoint capability cannot be utilized. Mbug supplies handlers for all exceptions and reports all exceptions encountered to the screen. Therefore, the user would be well advised to also copy any other vectors for which a

handler is not provided by the user.

The stack pointer (r15) is initialized to point at the the last longword at the top (highest address) in RAM.

All other user registers are reset to zero.

D.2.7 Software Watchdog

The software watchdog is disabled by Mbug during the reset initialization sequence. The user may enable the watchdog during the execution of user code; however, if control is returned to Mbug without first doing a hardware reset the watchdog will interrupt monitor operation with a watchdog reset once the counter expires.

D.2.8 Time of Day

The Time of Day peripheral module is disabled during Mbug initialization.

D.2.9 Periodic Interrupt

The periodic interrupt module is disabled during Mbug initialization.

D.2.10 Interrupt Controller

The 4 UART interrupt sources are enabled through the interrupt controller to the CPUs Normal interrupt input, although the interrupts are not currently turned on back in the UART. All other interrupt sources are blocked by the interrupt controller.

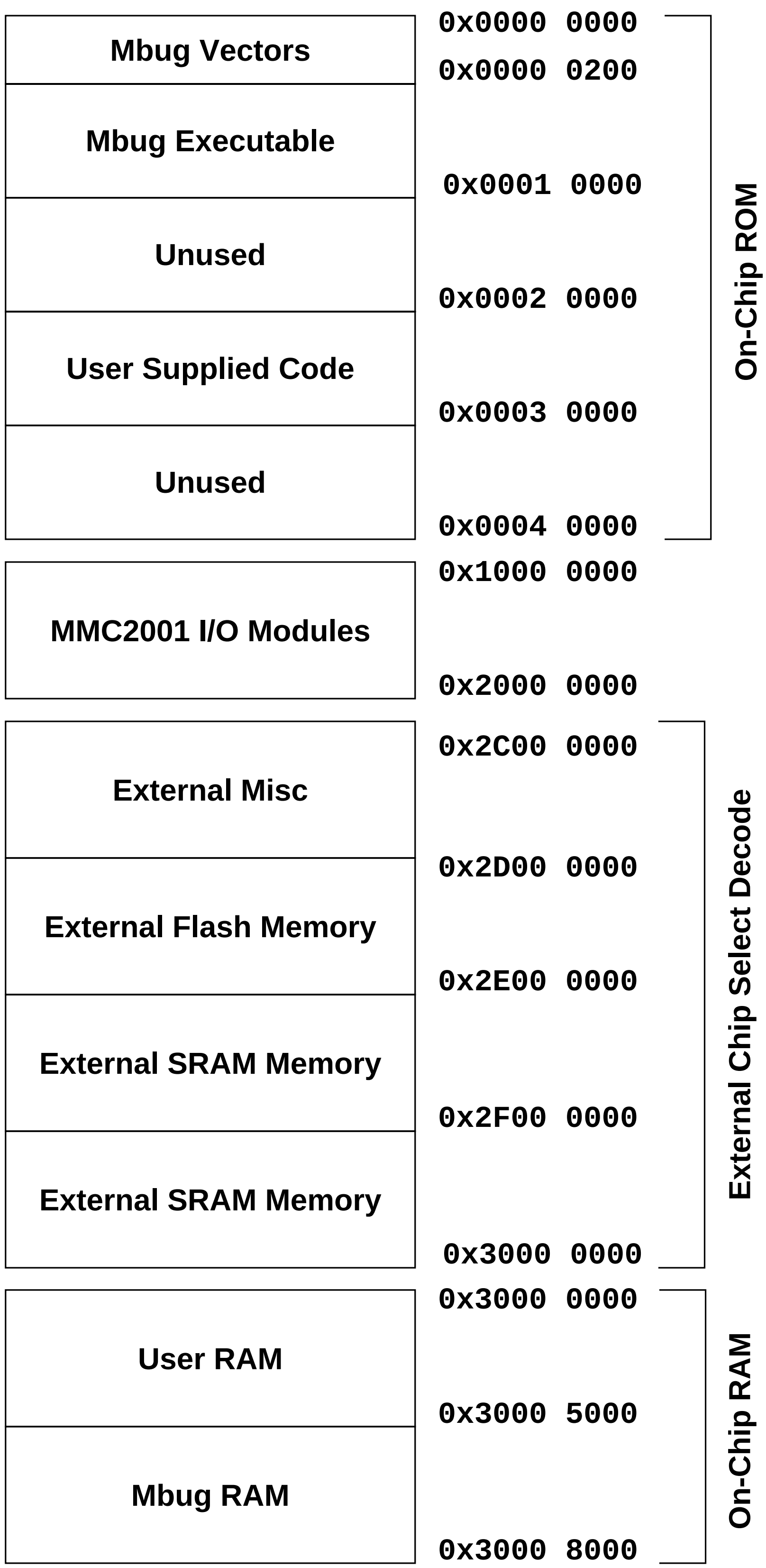
D.2.11 Serial Ports

Both UARTs are initialized by Mbug. UART 0 (Base address 0x10009000) is configured as the Mbug Keyboard port for 19200 baud at 32 MHz high-ref clock input (9600 at 16/16.367 MHz), 8 data bits, 2 stop bits, Rx and Tx interrupts disabled, and hardware handshaking enabled. UART1 (Base address 0x1000a000) is similarily configured with the exception of hardware handshaking and is used as the Host port by Mbug.

D.2.12 Other Peripherals

The remaining MMC2001 peripherals are not configured during Mbug initialization.

Figure 2-1. Mbug Memory Model (MMC2001Internal ROM)



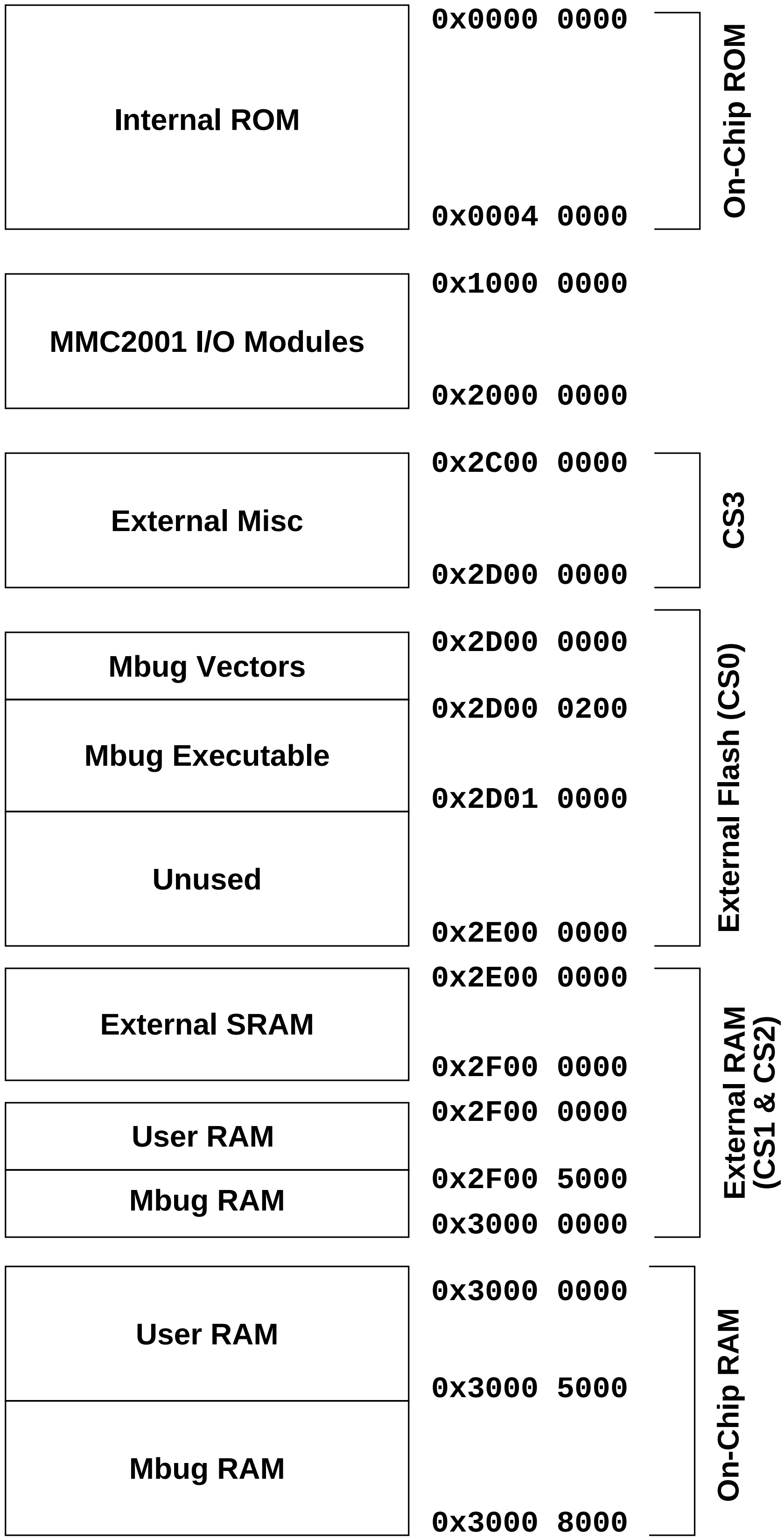


Figure 2-2Mbug Memory Model (CMB1200 Evaluation System)

Appendix E : : ASCII 7-Bit Character Set

MS LS	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0 0000	NUL	DLE	SP	0	@	P	‘	p
1 0001	SOH	DC1	!	1	A	Q	a	q
2 0010	STX	DC2	“	2	B	R	b	r
3 0011	ETX	DC3	#	3	C	S	c	s
4 0100	EOT	DC4	$	4	D	T	d	t
5 0101	ENQ	NAK	%	5	E	U	e	u
6 0110	ACK	SYN	&	6	F	V	f	v
7 0111	BEL	ETB	‘	7	G	W	g	w
8 1000	BS	CAN	(	8	H	X	h	x
9 1001	HT	EM	)	9	I	Y	i	y
A 1010	LF	SUB	*	:	J	Z	j	z
B 1011	VT	ESC	+	;	K	[	k	{
C 1100	FF	FS	,	<	L	\	l	|
D 1101	CR	GS	-	=	M	]	m	}
E 1110	SO	RS	.	>	N	^	n	~
F 1111	SI	US	/	?	O	_	o	DEL

Appendix F : Release Notes

Each release of the debugger firmware is documented below. Numbers in parenthesis refer to the number assigned to the anomoly in the bug/eratta database.

F.1 Revision 1.0

This version was released in the internal ROM on the MMC2001 (Powerstrike) Revision 0 silicon. Off-chip hardware configuration is consistent with the CMB1200 Evaluation System.

F.2 Revision 1.1

Internal development version only. Not formally released. This version fixes the problems identified in Version 1.0: PSR updates on Go and Trace commands (1), and assembly of DIVU instructions with a result register other than r1 (2). This version also contains numerous code rewrites for portability across platforms.

F.3 Revision 1.2

Initially released configured for the external flash on the CMB1200 Evaluation System. Initialization of VBR tracks the boot address starting in this version (4). It also corrects the reversed register fields on disassembly of load/store instructions (3).

F.4 Revision 1.3

Release configured for the external flash on the CMB1200 Evaluation System. Rewrote the way initialization of VBR tracks boot address. Changed port initialization to key only.

F.5 Revision 1.3.1

Code was not changed from 1.3, but link location of RAM has been changed to correspond to CS1 address range.

F.6 Revision 1.4

Internal development version only. Not formally released. Added disable of echo during downloads to fix recursion problem with I/O routines (6). Also added code to recover after monitor exceptions by jumping back to main loop (5).

F.7 Revision 1.5

Initially released configured for the external flash on the CMB1200 Evaluation System. Corrects problems with GO and TRACE commands requiring word aligned addresses (7). Adds warm start routine for gracefull recovery from exceptions during monitor operation (8). Adds new breakpoint command names and capability to

remove all breakpoints at once. Changed register modify command name to RM from RMM. All old commands still recognized. Updates help messages to document new commands.

F.8 Revision 1.6

Initially released configured for the external flash on the CMB1200 Evaluation System. Corrects remaining problems with GO and TRACE commands to halfword aligned addresses (9). Also fixes word opcode size problems in non-interactive assembler (as1) command (10). New Memory Display halfword and byte commands added (11). Help reformatted and long form commands deleted to reduce ROM footprint below 64k.